

FIG. 1

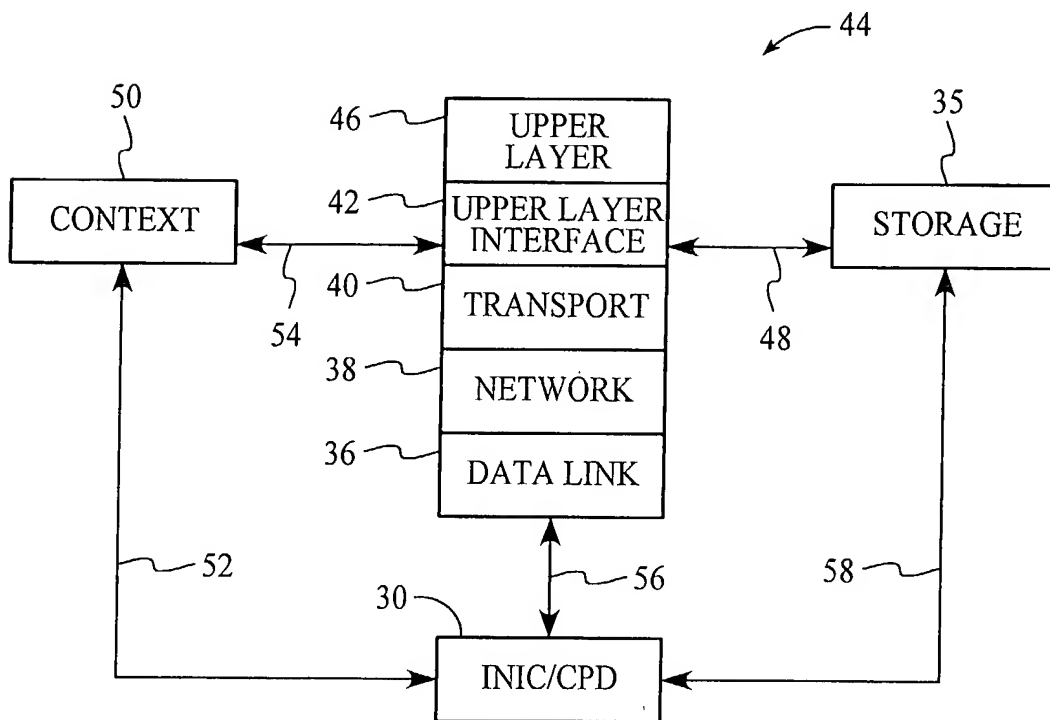


FIG. 2

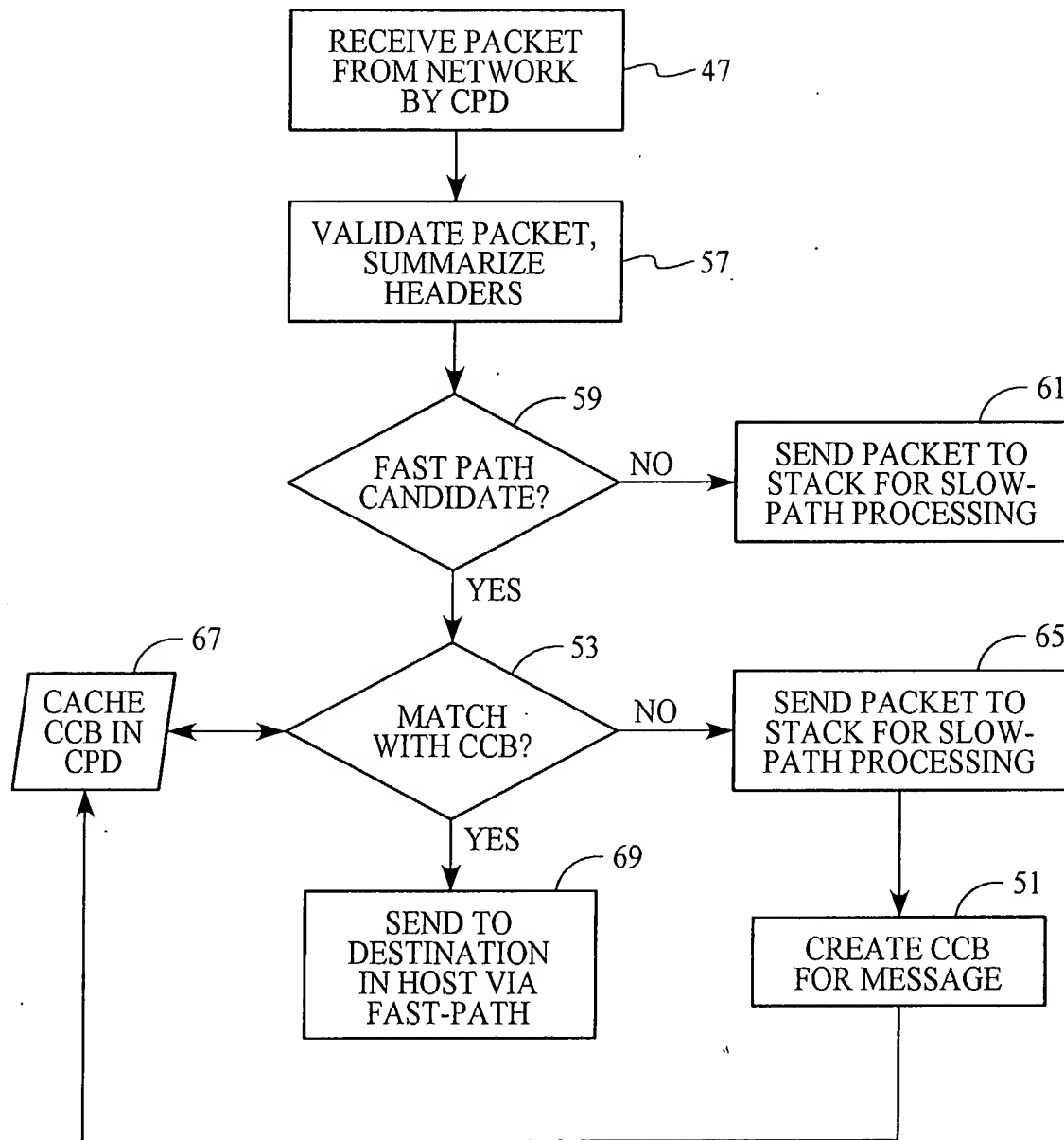


FIG. 3

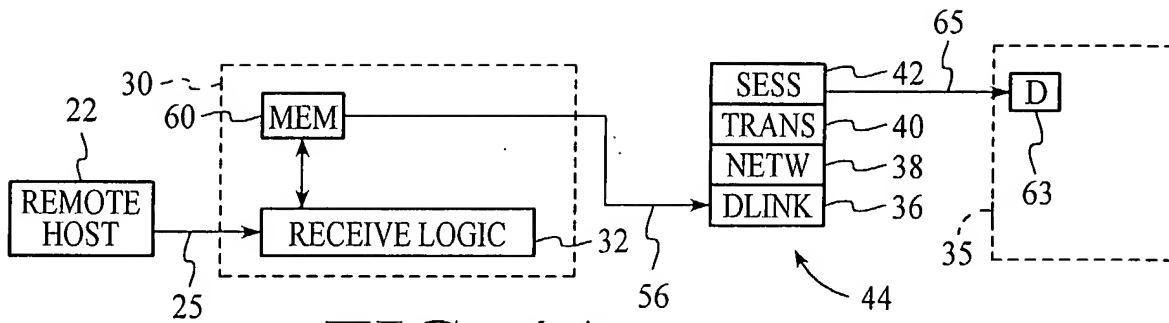


FIG. 4A

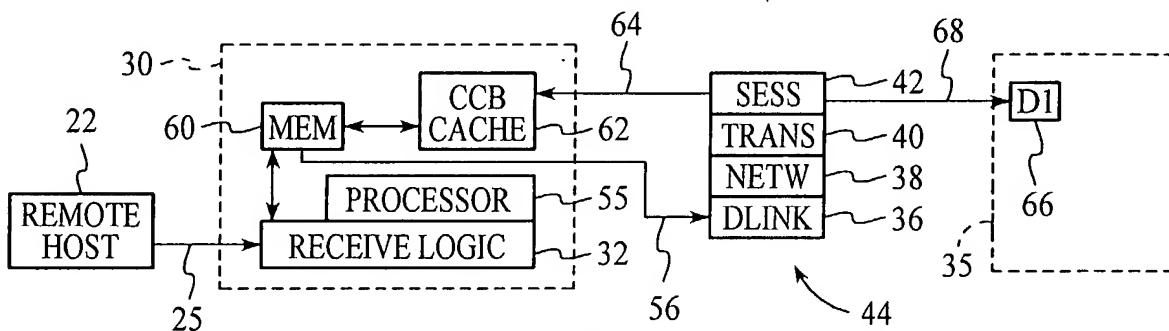


FIG. 4B

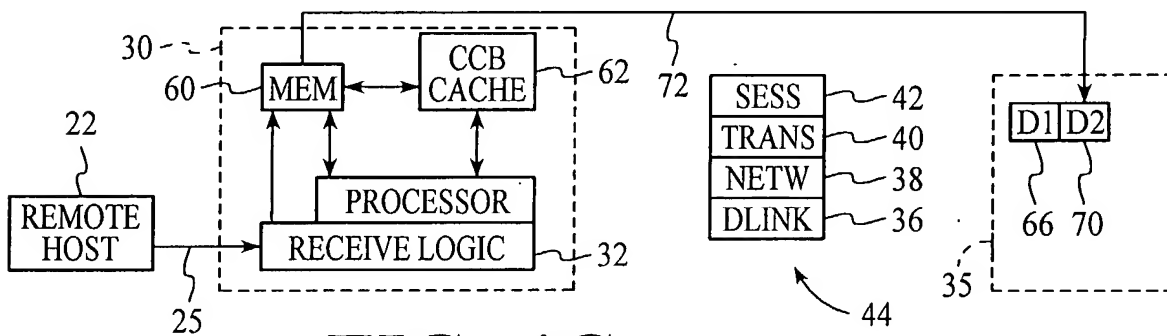


FIG. 4C

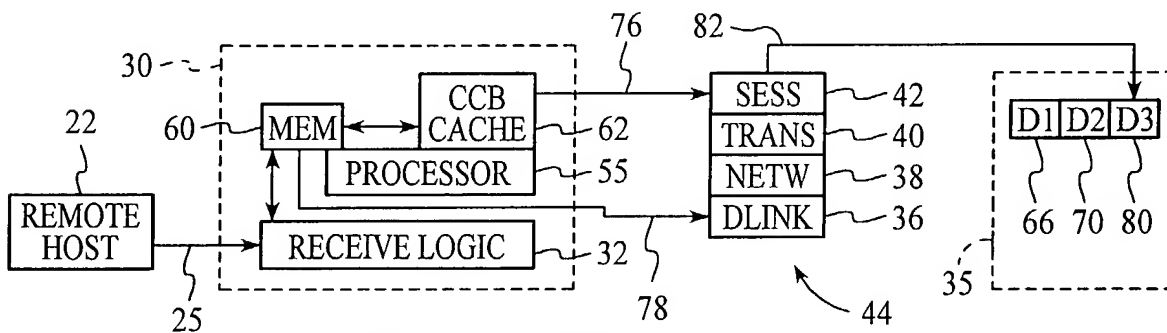


FIG. 4D

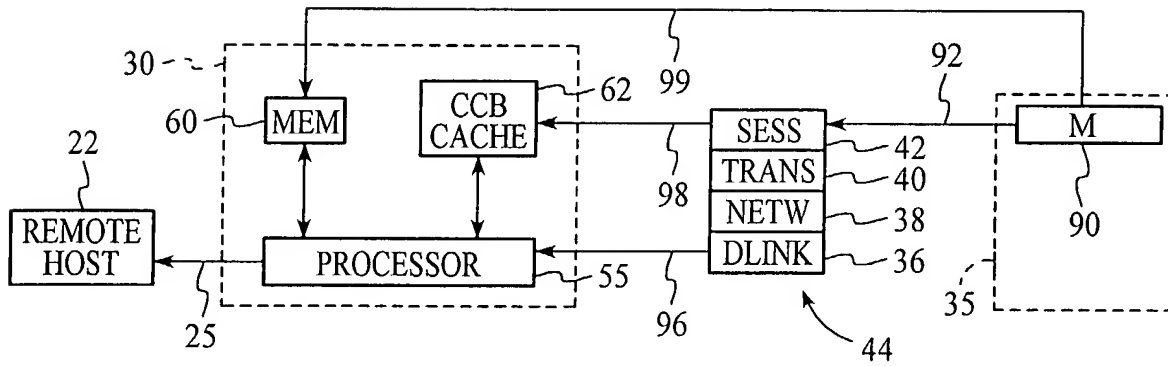


FIG. 5

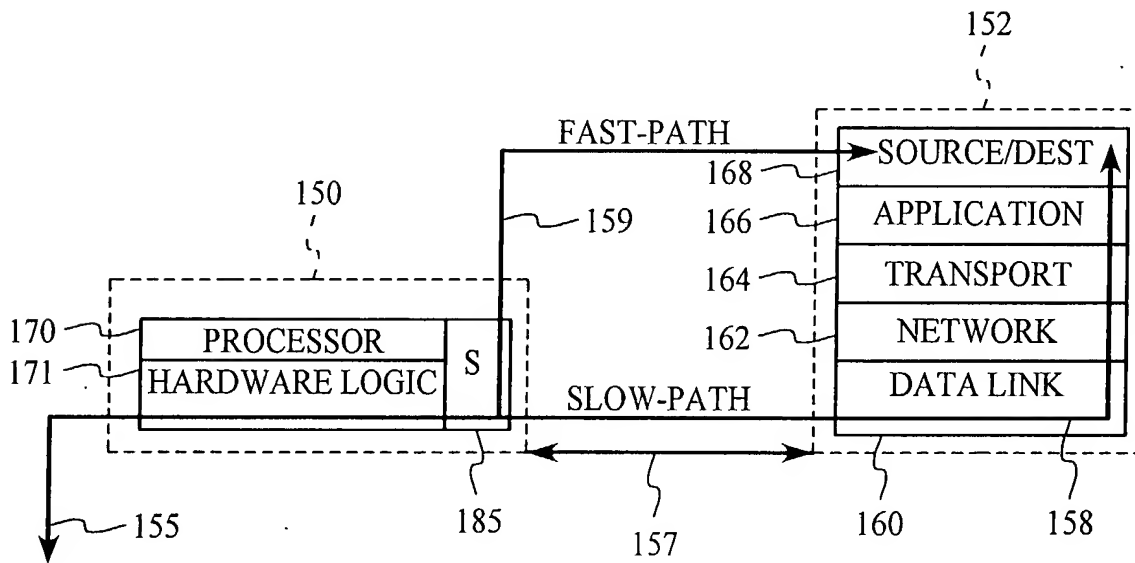


FIG. 6

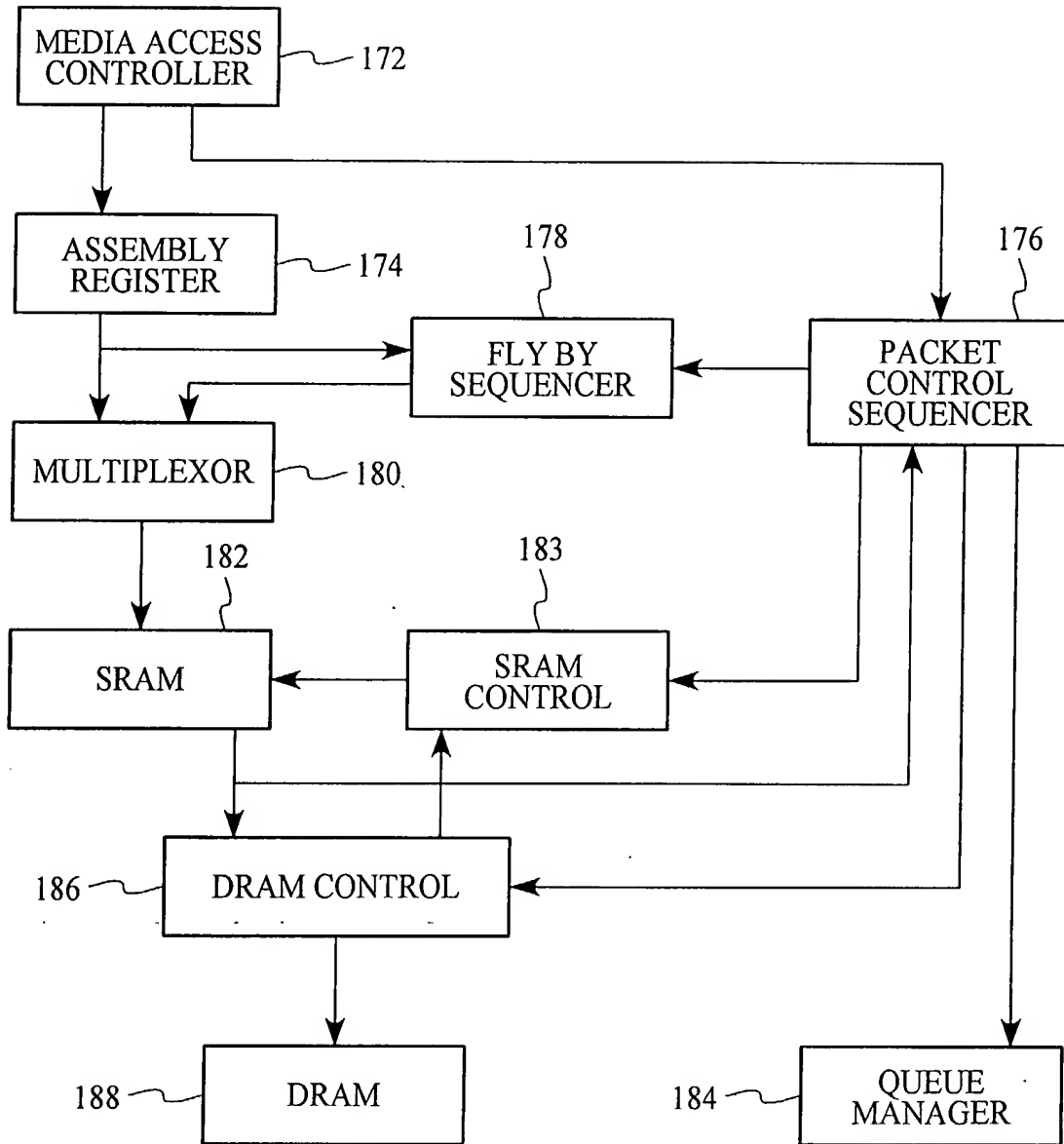


FIG. 7

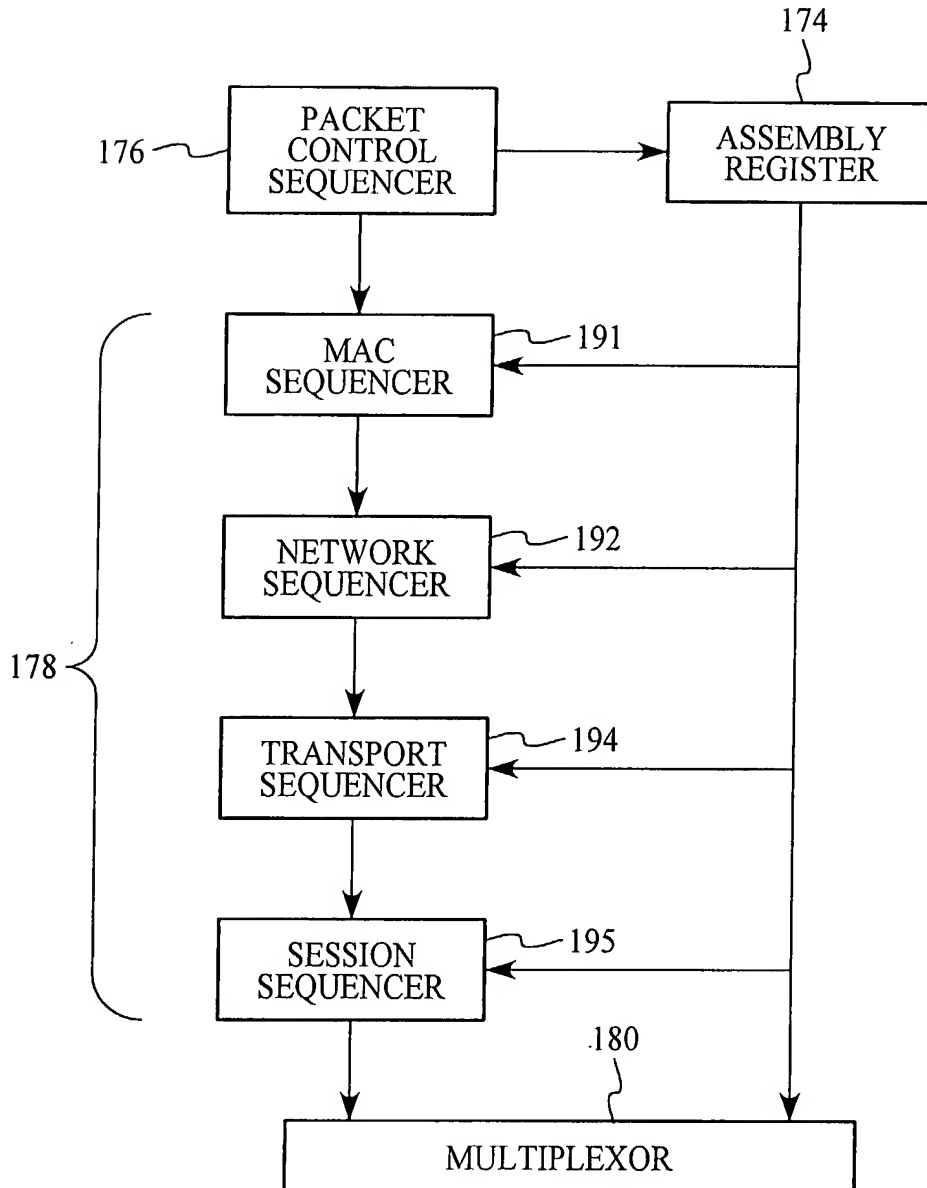


FIG. 8

7/82

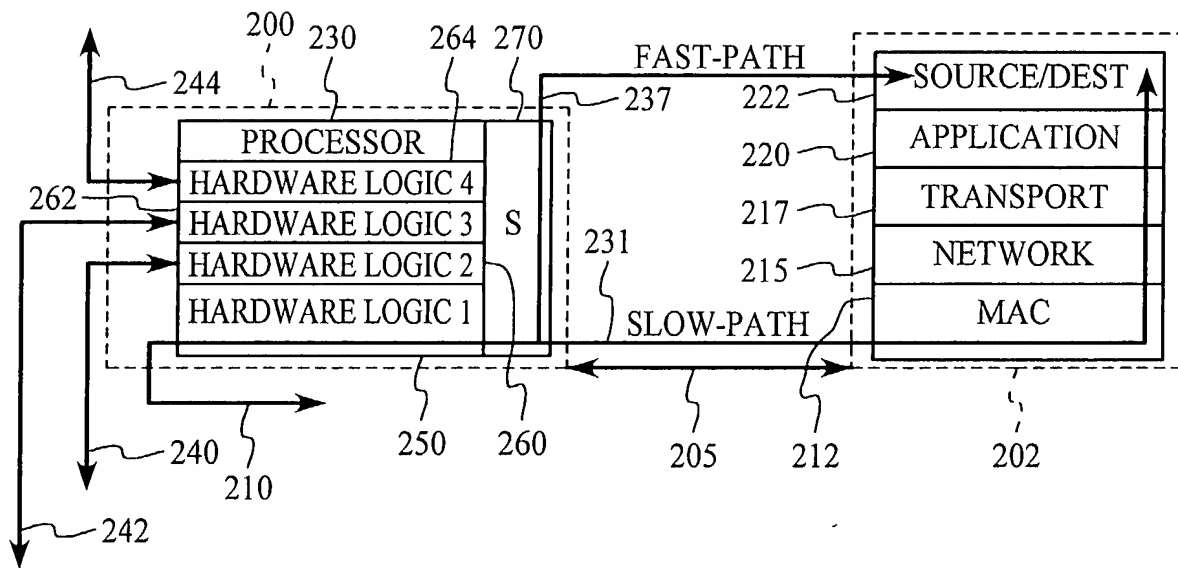


FIG. 9

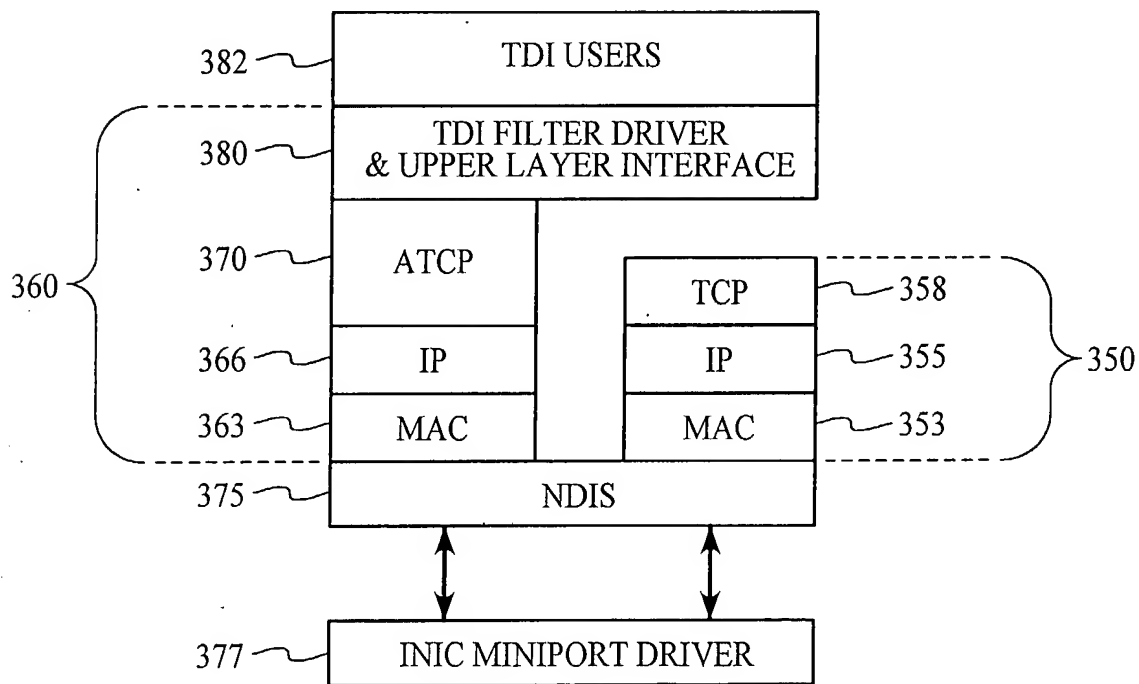


FIG. 11

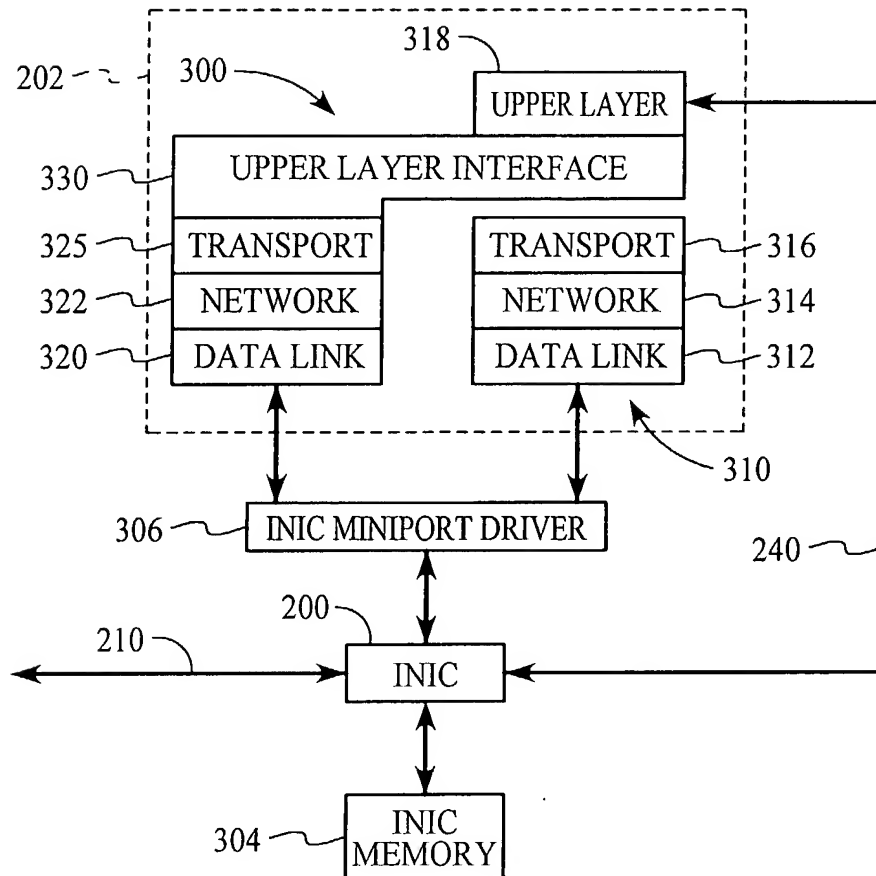


FIG. 10

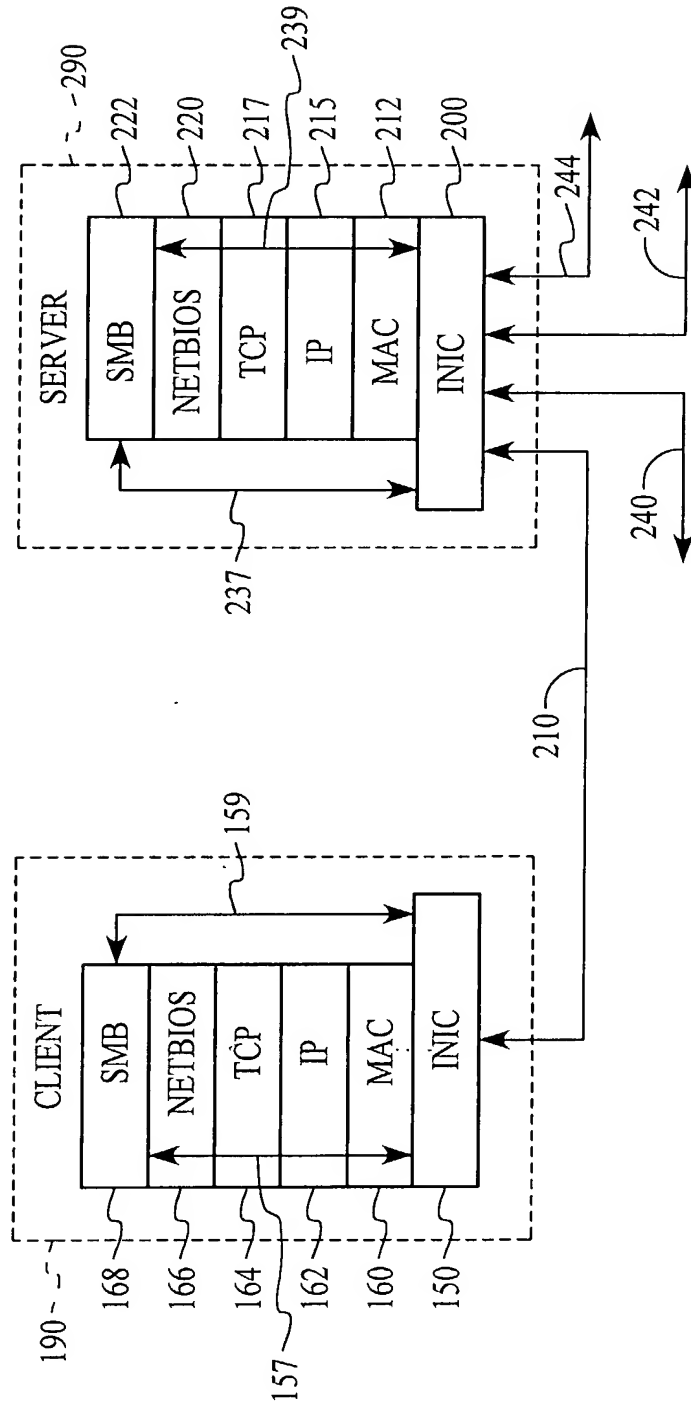


FIG. 12

10/82

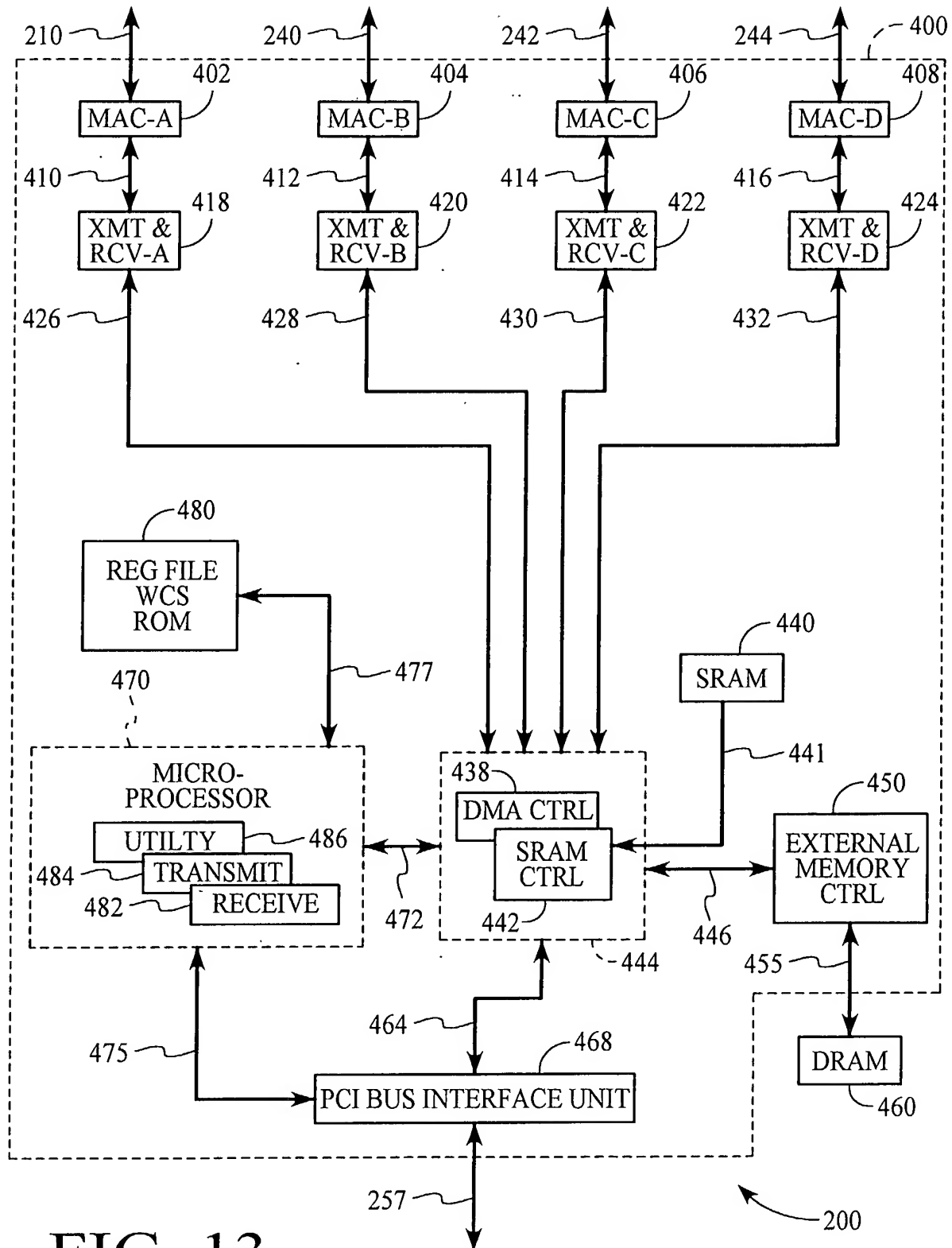


FIG. 13

11/82

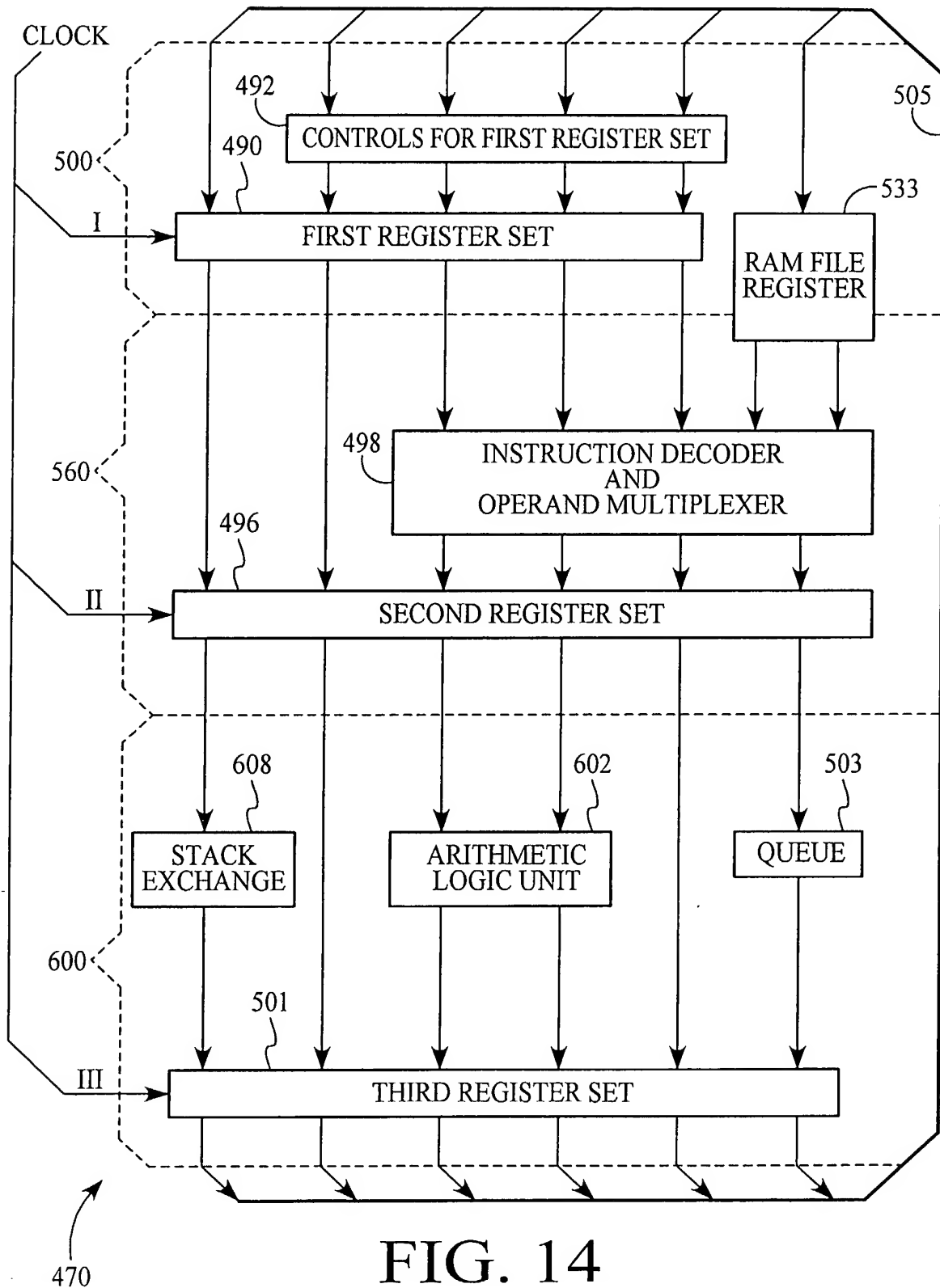


FIG. 14

12/82

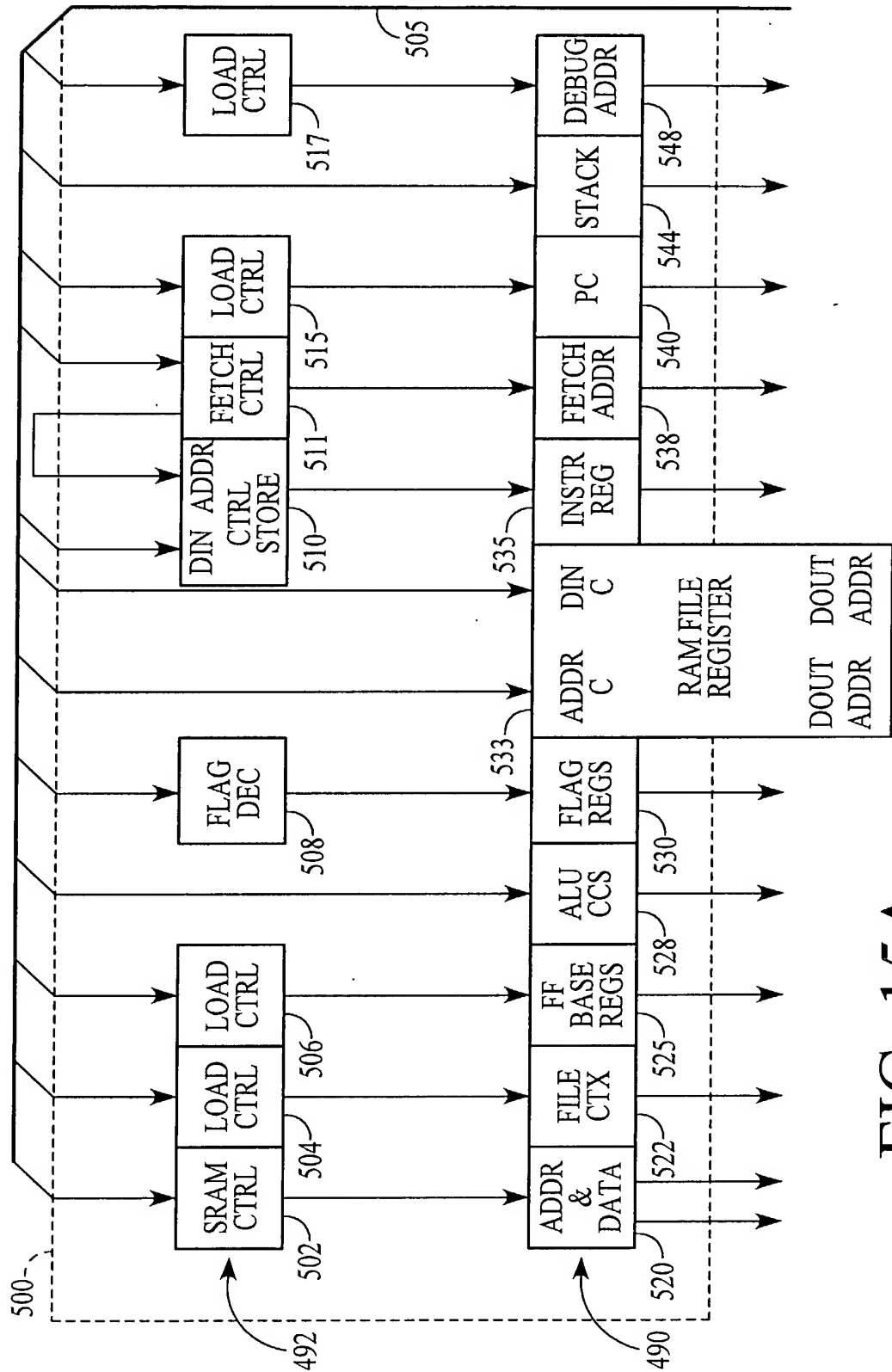


FIG. 15A

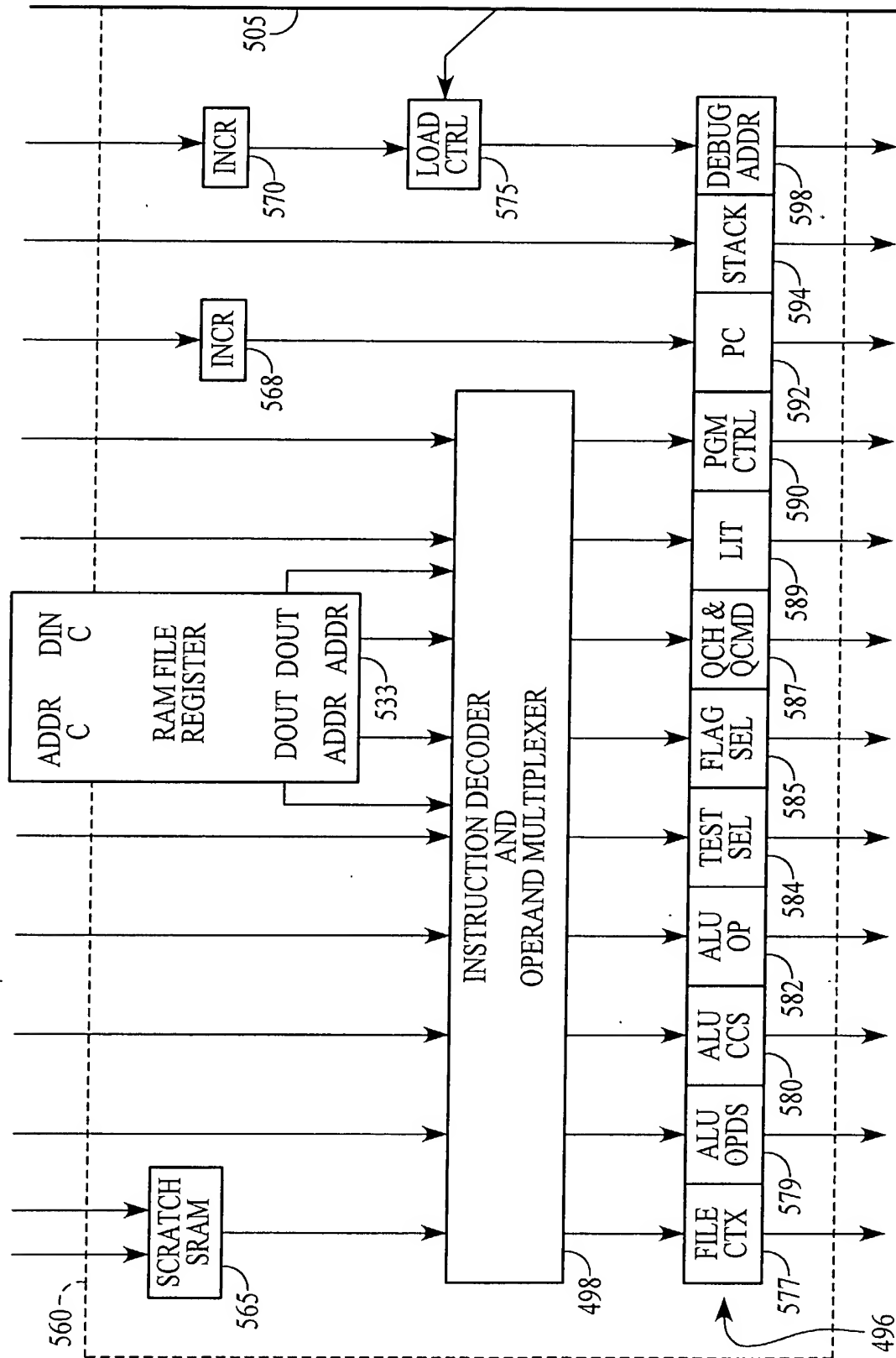


FIG. 15B

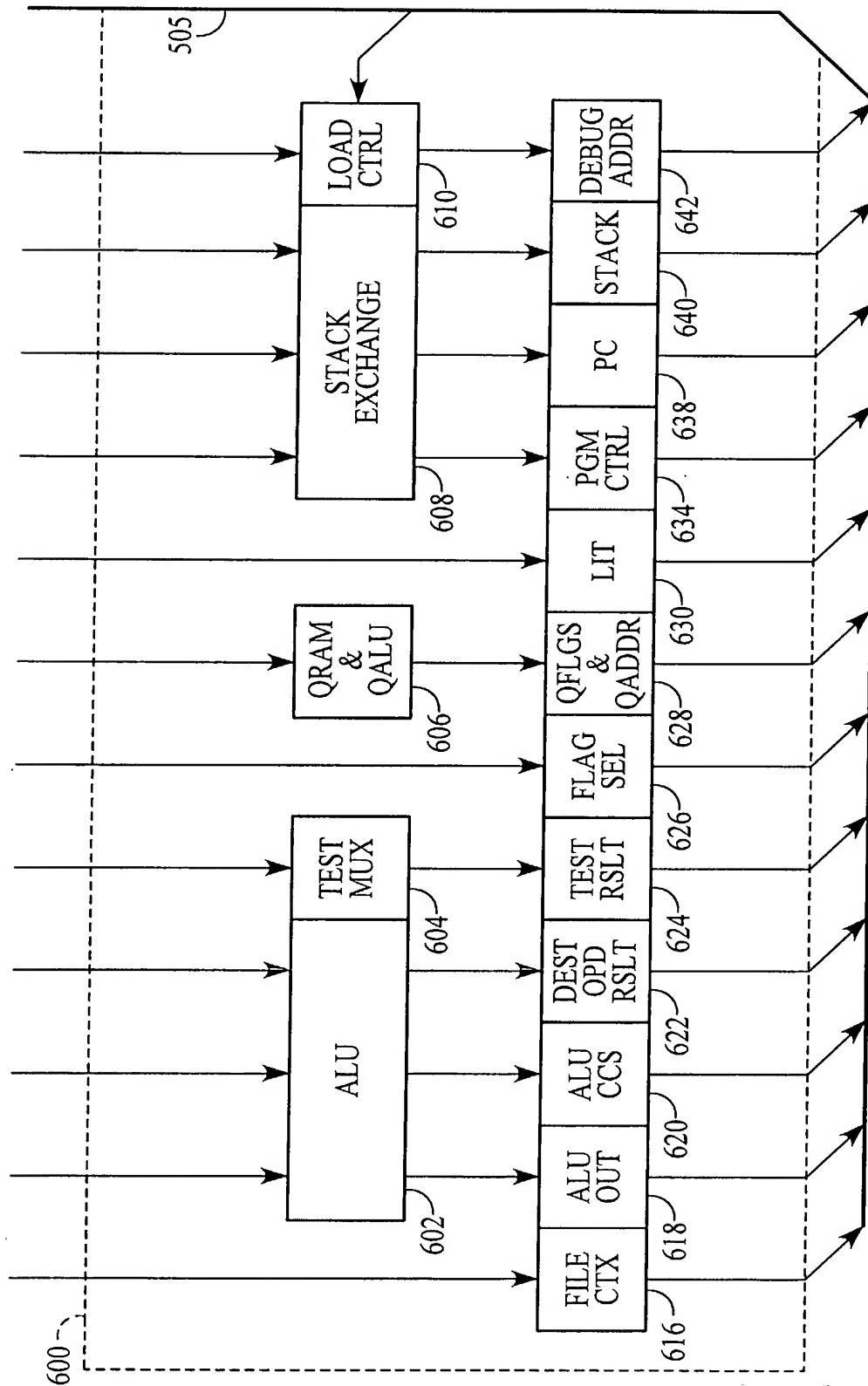


FIG. 15C

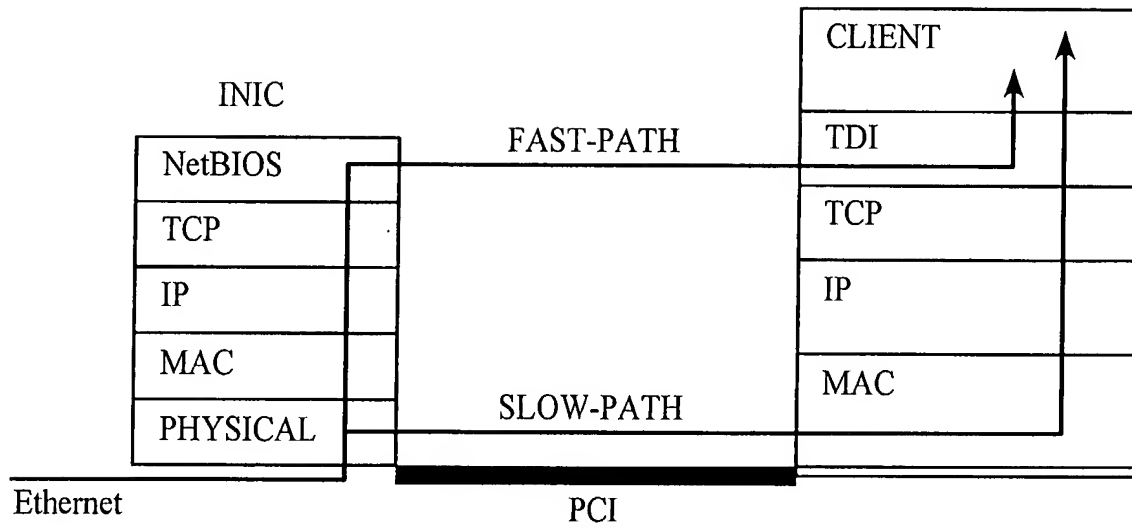


FIG. 16

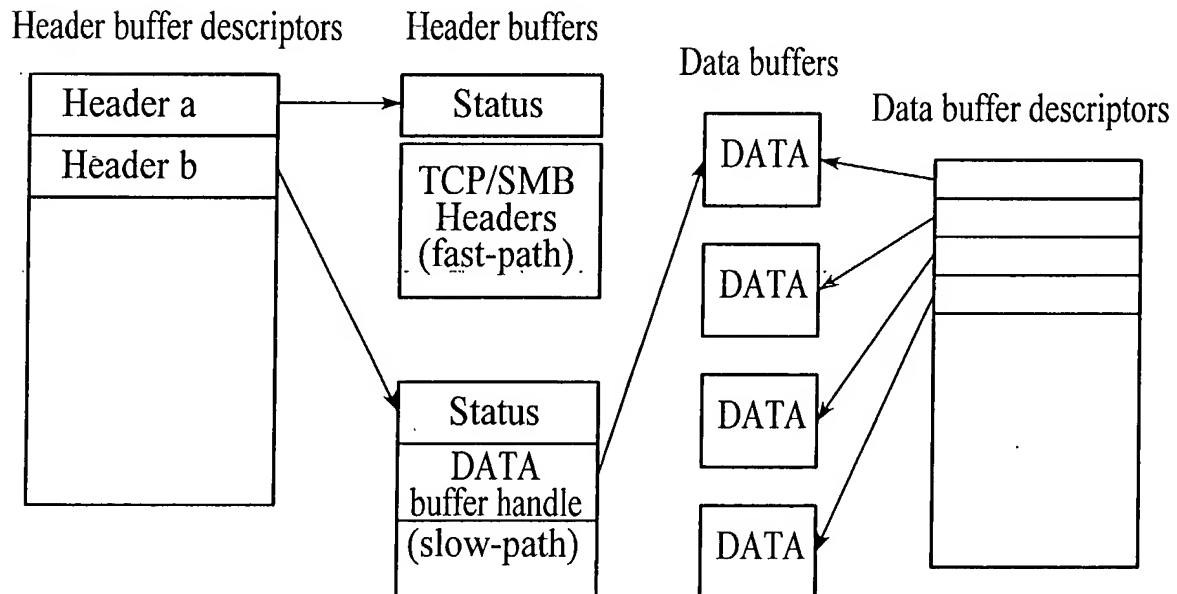


FIG. 17

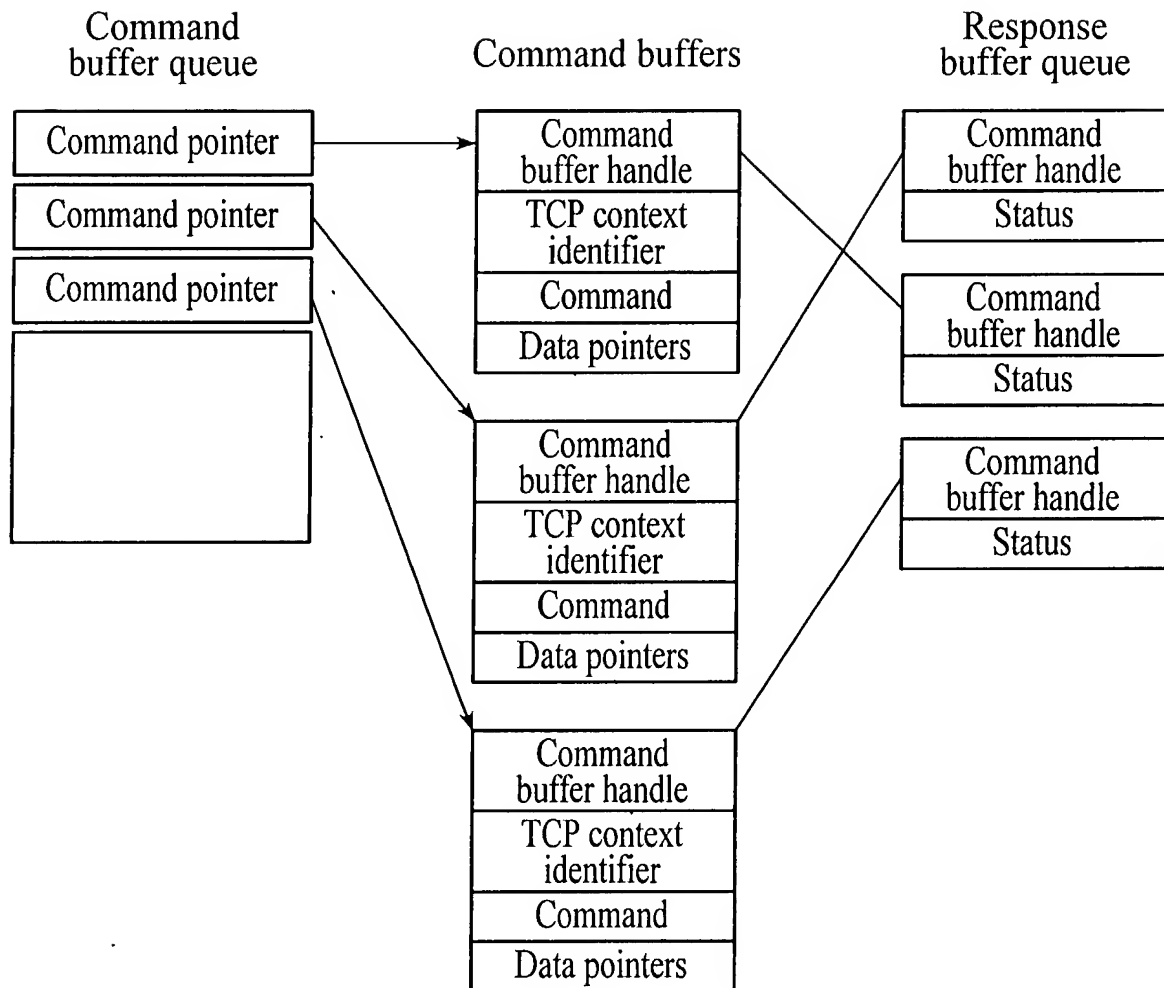


FIG. 18

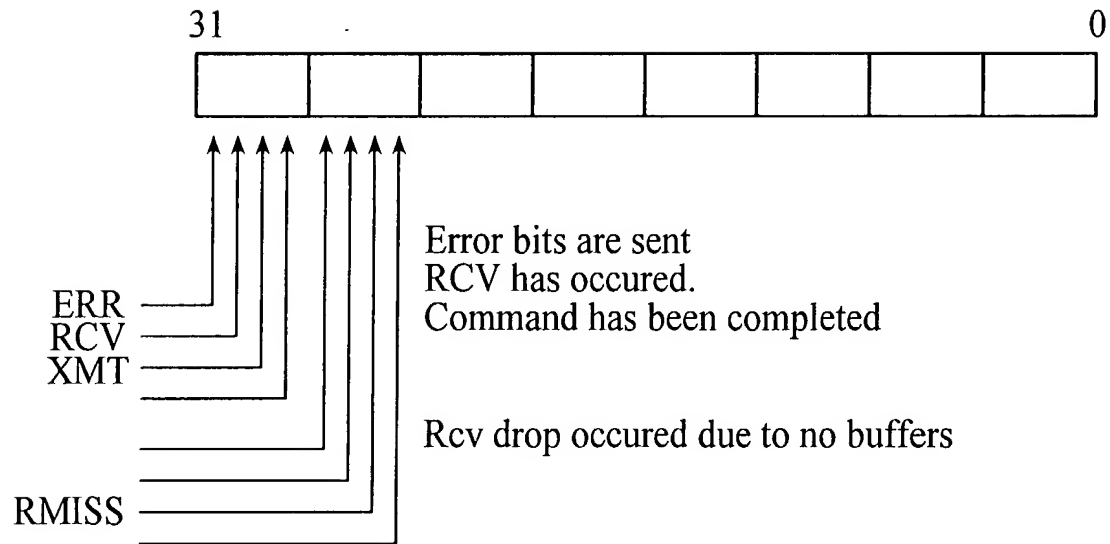


FIG. 19

ISR	0x0	Interrupt Status
IMR	0x4	Interrupt Mask
HBAR	0x8	Header Buffer Address
DBHR	0xC	Data Buffer Handle
DBAR	0x10	Data Buffer Address
CBAR0	0x14	Command Buffer Address XMT0
CBAR1	0x18	Command Buffer Address XMT1
CBAR2	0x1C	Command Buffer Address XMT2
CBAR3	0x20	Command Buffer Address XMT3
CBAR4	0x24	Command Buffer Address RCV
RBAR	0x28	Response Buffer Address

FIG. 20

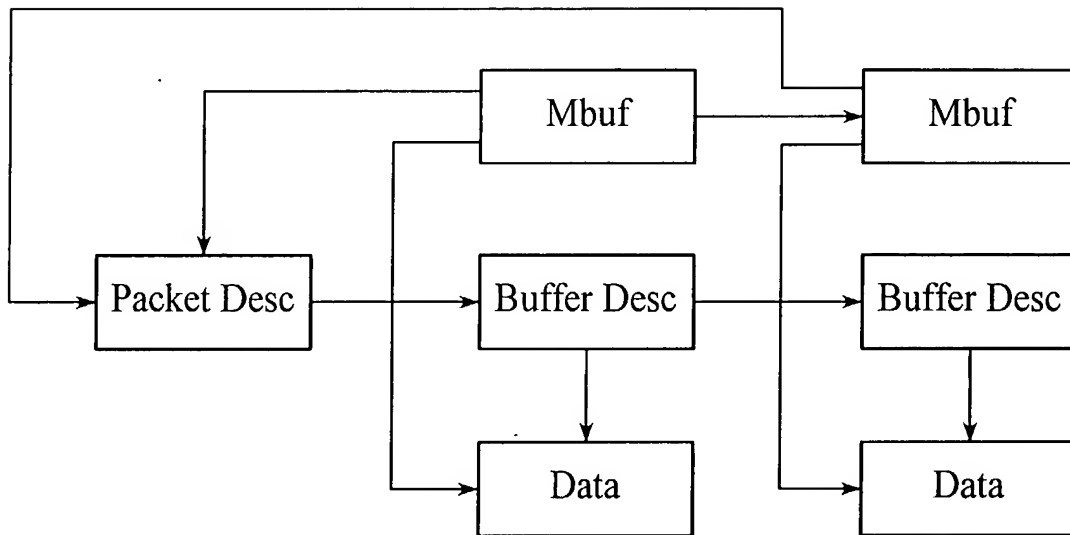


FIG. 21

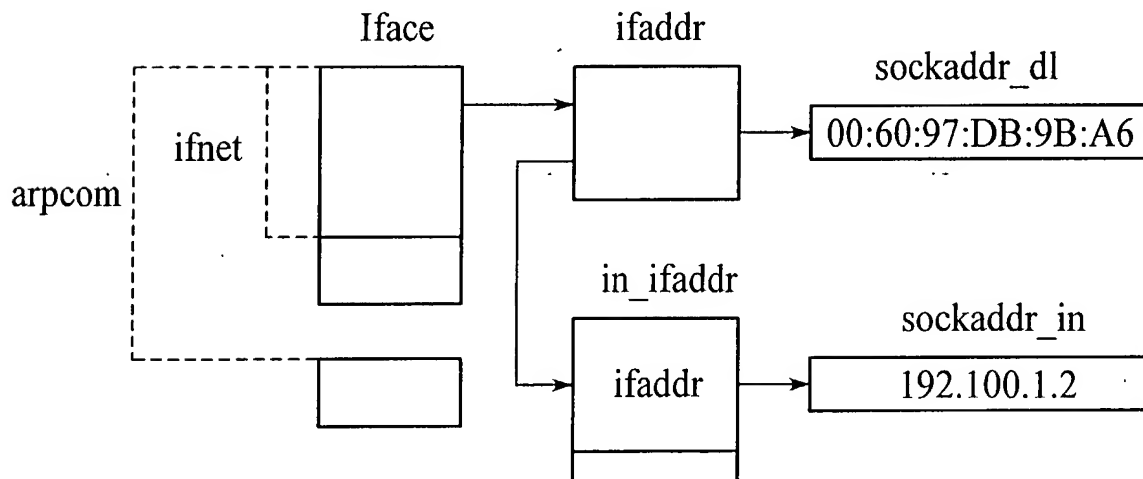


FIG. 22

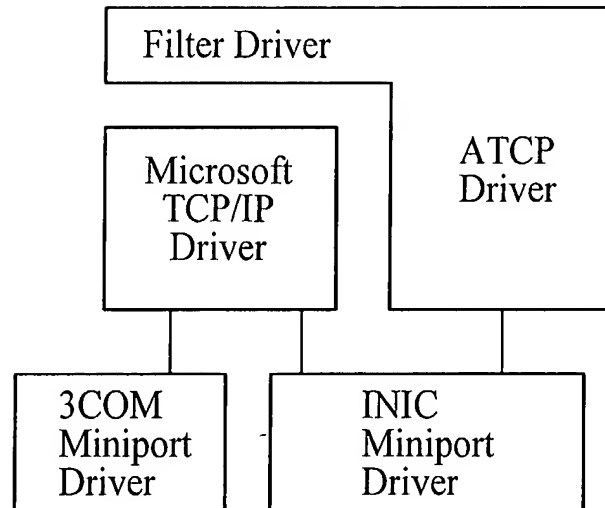
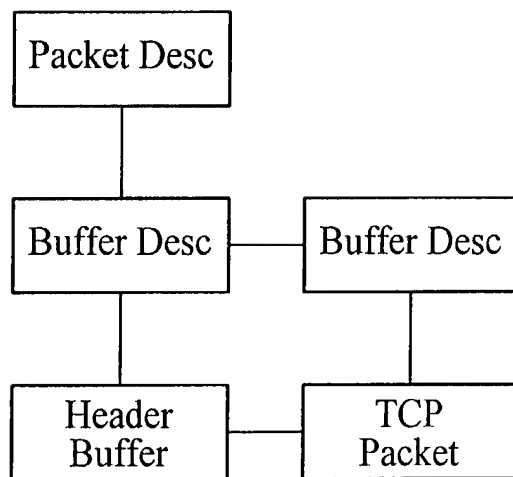
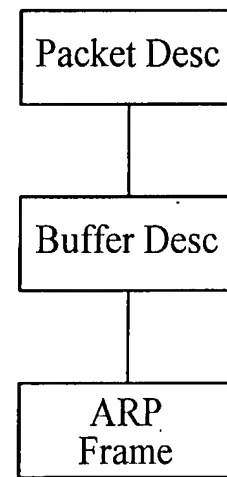


FIG. 23



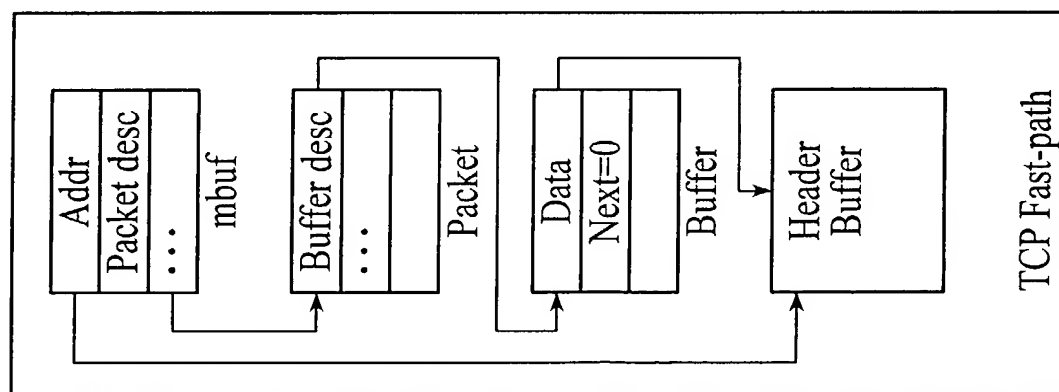
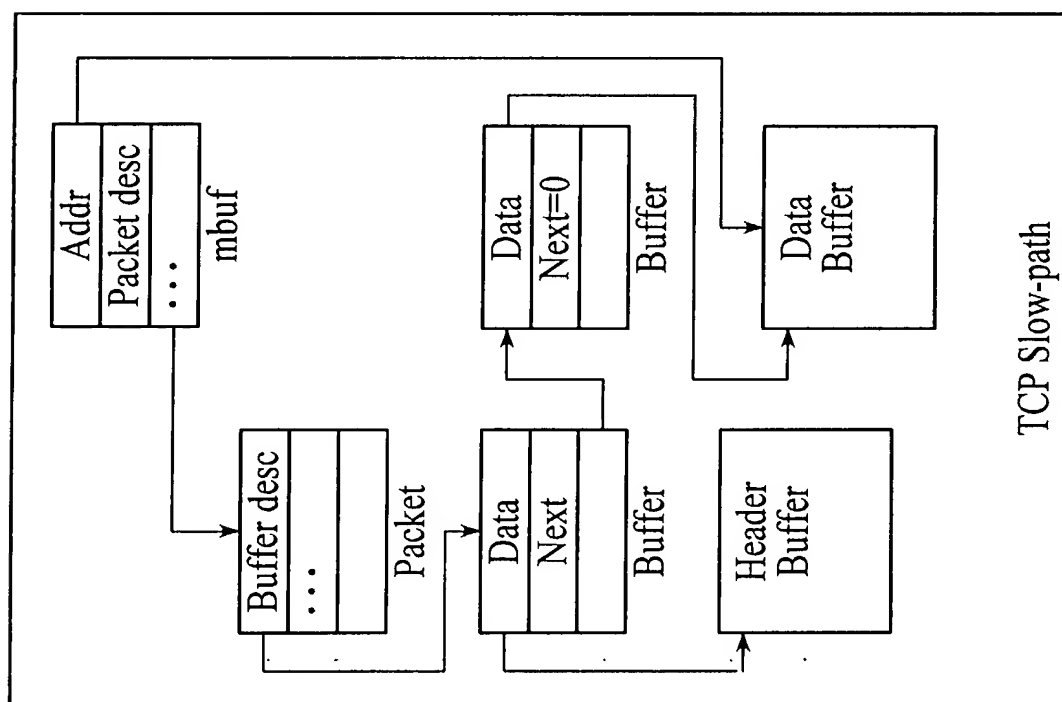
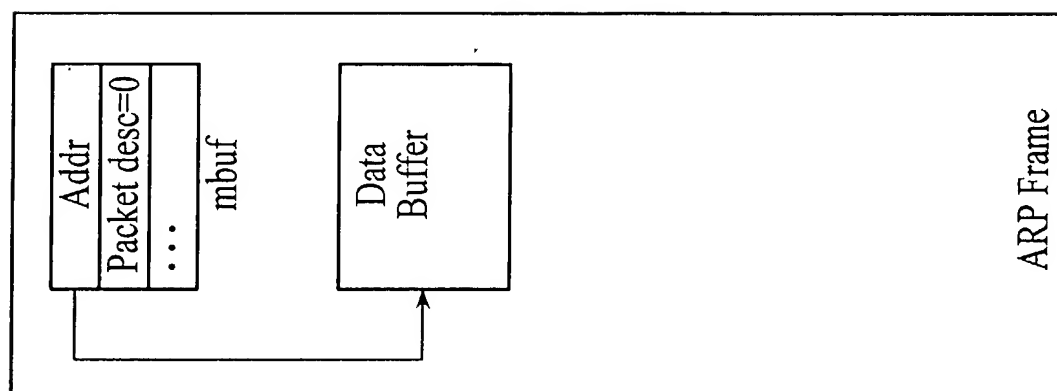
Example of incoming TCP pkt

FIG. 24



Example of incoming ARP Frame

FIG. 25



21/82

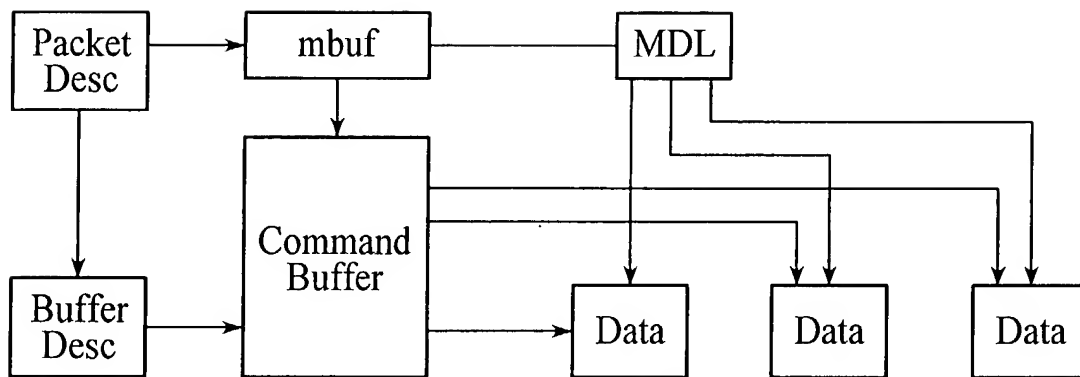


FIG. 27

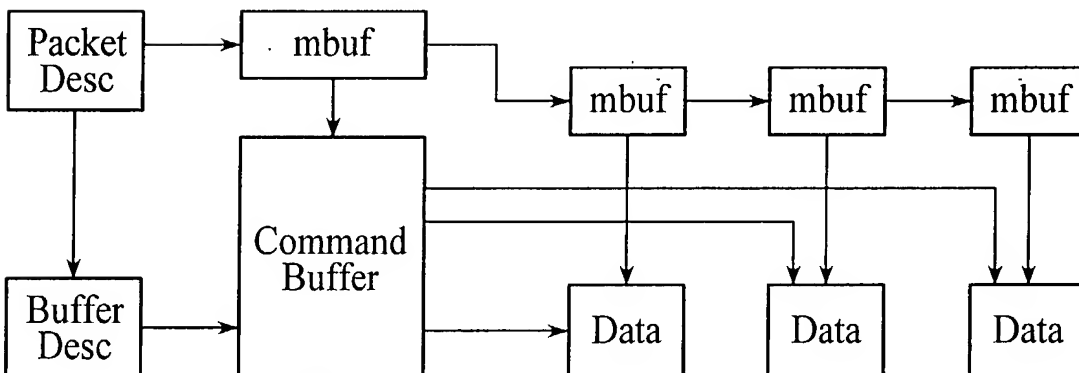


FIG. 28

22/82

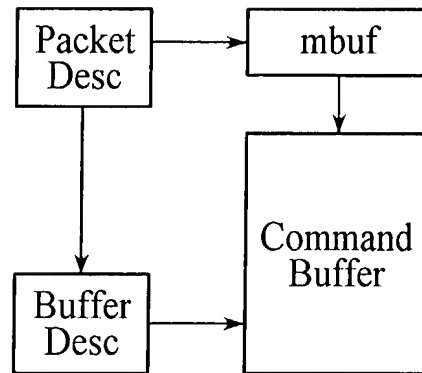


FIG. 29

SRAM requirements for the Receive and Transmit engines:

TCB buffers	256 bytes* 16	4096
Header buffers	128 bytes* 16	2048
TCB hash index	16 bytes* 256	4096
Timers		128
DRAM Fifo queues	128 bytes* 16	<u>2048</u>
		~12K bytes

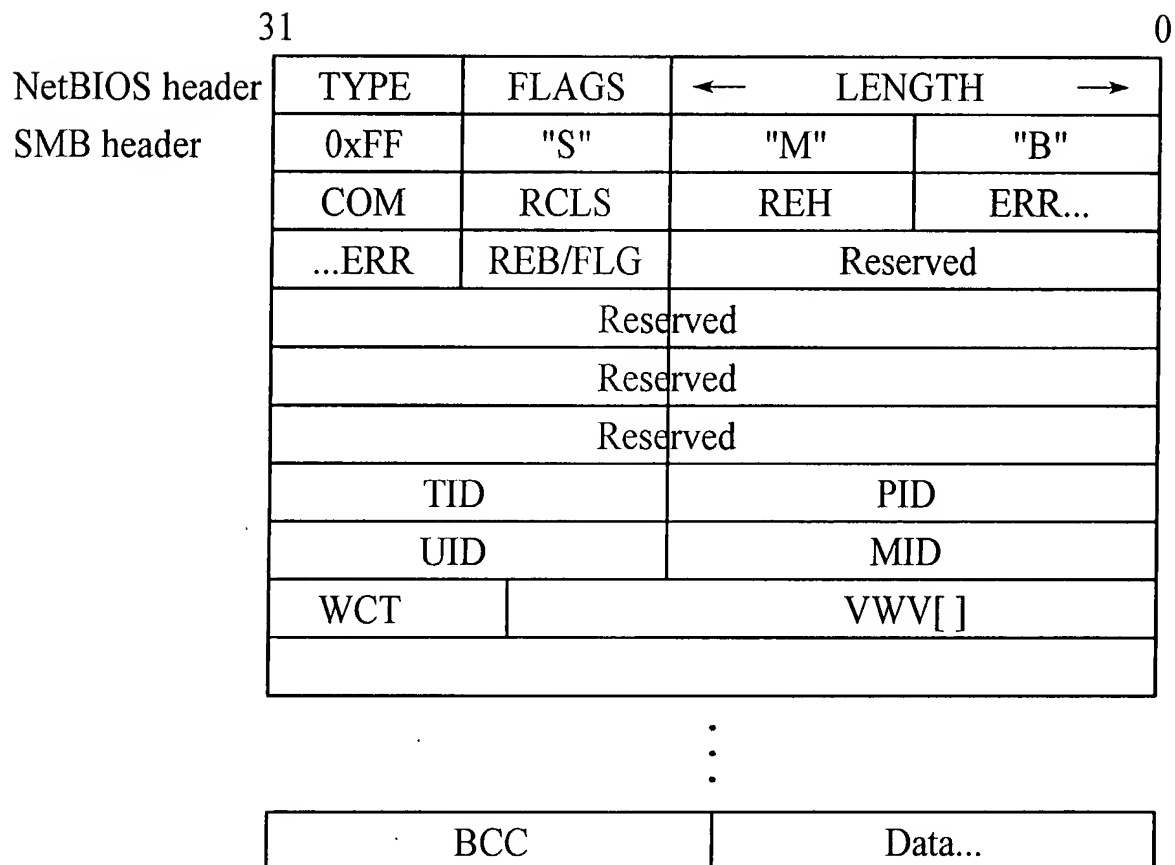
FIG. 30

Summary of the main loop of Receive:

```
forever {  
  while there are any Receive events {  
    if (a new event) {  
      if (no new context available)  
        ignore the event;  
    }  
    call appropriate event handler to service the event;  
    this may make a waiting process runnable or set up  
    a new process to be run (get free context, hddr buffer,  
    TCB buffer, set the context up).  
  }  
  while any process contexts are runnable {  
    run them by jumping to the start/resume address;  
    if (process complete)  
      free the context;  
  }  
}
```

FIG. 31

Format of the SMB header of an SMB frame:



Notes (interesting fields):

LENGTH 17 bit Length of SMB message (0 - 128K)

COM SMB command

WCT Count (16 bit) of parameter words in VWV []

VWV Variable number of parameter words

BCC Bytes of data following

FIG. 32

25/82

Summary of the main loop of Transmit:

```
forever {  
    while there are any Transmit events {  
        if (a new event) {  
            if (no new context available)  
                ignore the event;  
        }  
        call appropriate event handler to service the event;  
        this may make a waiting process runnable or set up  
        a new process to be run (get free context, hddr buffer,  
        TCB buffer, set the context up).  
    }  
    while any process contexts are runnable {  
        run them by jumping to the start/resume address;  
        if (process complete)  
            free the context;  
    }  
}
```

**FIG. 33**

Bit 31 - 24 Byte enable 7 - 0. Only the low order four bits are
 valid for 32 bit addressing mode.
 Bit 23 - 0 Memory access
 1 Configuration access
 Bit 22 - 0 Read (to Host)
 1 Write (to Host)
 Bit 21 - 1 Data Valid
 Bit 20 - 16 Reserved
 Bit 15 - 0 Address



FIG. 34

Configuration Space 1

00
 04
 08
 0C
 10
 3C

SRAM Address Offset

00
 04
 08
 0C
 10
 14

Configuration Space 2

00
 04
 08
 0C
 10
 3C

00
 18
 08
 1C
 20
 24

All other reads to configuration space will return 00.

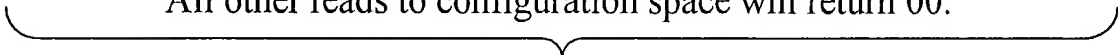


FIG. 35

27/82

- Bit 0 - 0 I/O accesses are not enabled
- Bit 1 - 1 Memory accesses are enabled
- Bit 2 - 1 Bus master is enabled
- Bit 3 - 0 Special Cycle is not enabled
- Bit 4 - 1 Memory Write and Invalidate is enabled
- Bit 5 - 0 VGA palette snooping is not enabled
- Bit 6 - 1 Parity checking is enabled
- Bit 7 - 0 Address data stepping is not enabled
- Bit 8 - SERR# is enabled
- Bit 9 - 0 Fast back to back is not enabled

FIG. 36

- Bit 5 - 1 66 MHz capable is enabled. This bit will be set if the INIC
Detects the system running at 66 MHz on reset
- Bit 6 - 0 User Definable Features is not enabled
- Bit 7 - 1 Fast Back-to-Back slave transfers enabled
- Bit 8 - 1 Parity Error enabled - This bit is initialized to 0
- Bit 9,10 - 00 - Fast device select will be set if we are at 33 MHz
01 - Medium device select will be set if we are at 66 MHz
- Bit 11 - 1 Target Abort is implemented. Initialized to 0.
- Bit 12 - 1 Target Abort is implemented. Initialized to 0.
- Bit 13 - 1 Master Abort is implemented. Initialized to 0.
- Bit 14 - 1 SERR# is implemented. Initialized to 0.
- Bit 15 - 1 Parity error is implemented. Initialized to 0.

FIG. 37

28/82

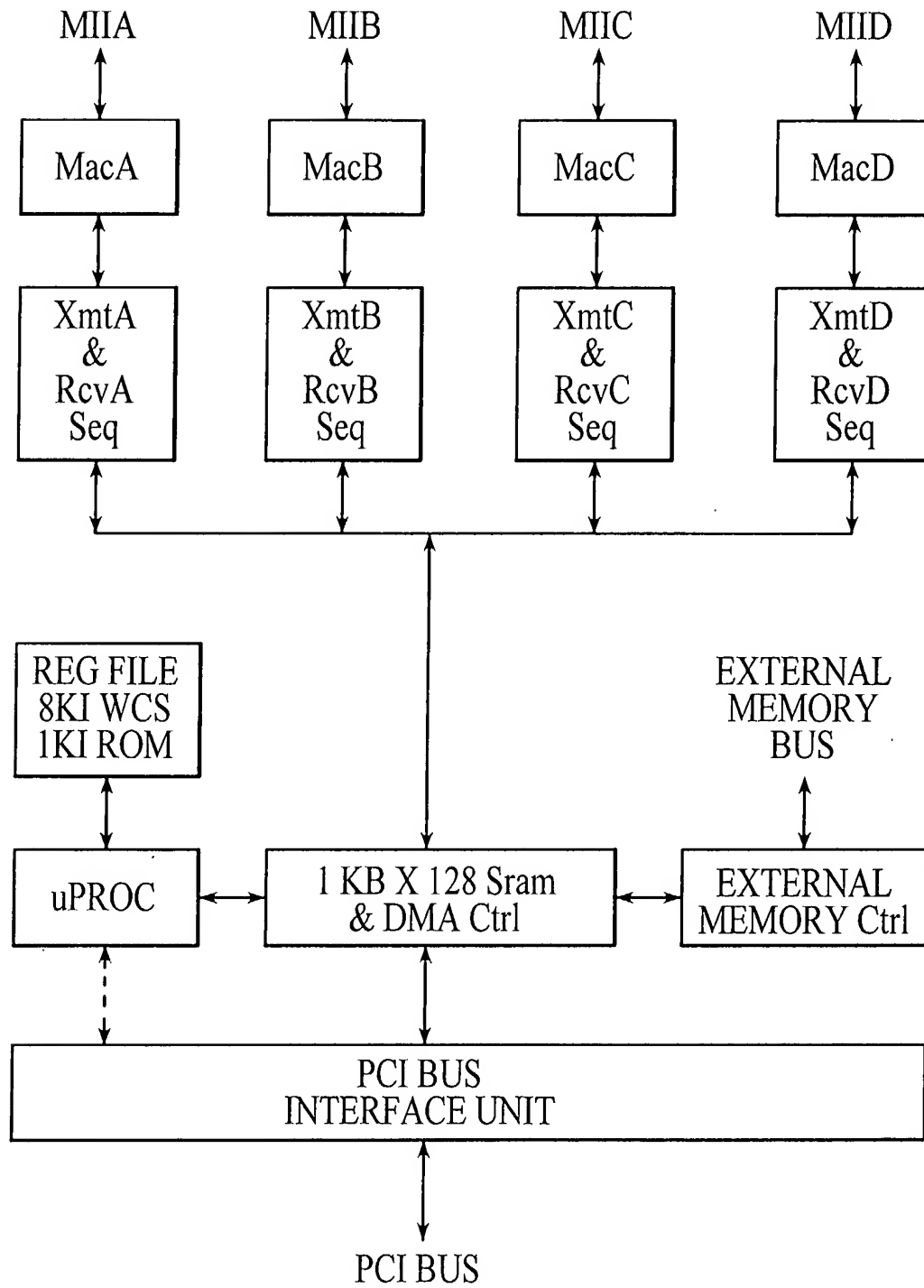


FIG. 38

29/82

<u>MODULE</u>	<u>DESCR</u>	<u>SPEED</u>	<u>AREA</u>
Scratch RAM,	1Kx128 sport,	4.37 ns nom.,	06.77 mm ²
WCS,	8Kx49 sport,	6.40 ns nom.,	18.29 mm ²
MAP,	128x7 sport,	3.50 ns nom.,	00.24 mm ²
ROM,	1Kx49 32col,	5.00 ns nom.,	00.45 mm ²
REGs,	512x32 tport,	6.10 ns nom.,	03.49 mm ²
Macs,	.75 mm ² x 4 =		03.30 mm ²
PLL,	.5 mm ² =		00.55 mm ²
MISC LOGIC,	117,260 gates / (5035 gates / mm ²) =		23.29 mm ²
TOTAL CORE			56.22 mm ²
(Core side) ²		=	56.22 mm ²
Core side		=	07.50 mm
Die side	= core side + 1.0 mm (I/O cells)	=	08.50 mm
Die area	= 8.5 mm x 8.5 mm	=	72.25 mm ²
Pads needed	= 220 signals x 1.25 (vss, vdd)	=	275 pins
LSI PBGA		=	272 pins



FIG. 39

(10MB/s/100Base) x 2 (full duplex) x 4 connections	= 80 MB/s
Average frame size	= 512 B
Frame rate = 80MB/s / 512B	= 156,250 frames / s
Cpu overhead / frame = (256B context read) + (64B header read) + (128B context write) + (128B misc.)	= 512B / frame
Total bandwidth = (512B in) + (512B out) + (512B Cpu)	= 1536B / frame
Dram Bandwidth required = (1536B/frame) x (156,250 frames/s)	= 240MB/s
Dram Bandwidth @ 60MHz = (32 bytes / 167ns)	= 202MB/s
Dram Bandwidth @ 66MHz = (32 bytes / 150ns)	= 224MB/s
PCI Bandwidth required	= 80MB/s
PCI Bandwidth available @ 30 MHz, 32b, average	= 46MB/s
PCI Bandwidth available @ 33 MHz, 32b, average	= 50MB/s
PCI Bandwidth available @ 60 MHz, 32b, average	= 92MB/s
PCI Bandwidth available @ 66 MHz, 32b, average	= 100MB/s
PCI Bandwidth available @ 30 MHz, 64b, average	= 92MB/s
PCI Bandwidth available @ 33 MHz, 64b, average	= 100MB/s
PCI Bandwidth available @ 60 MHz, 64b, average	= 184MB/s
PCI Bandwidth available @ 66 MHz, 64b, average	= 200MB/s

FIG. 40

Receive frame interval = 512B / 40MB/s	= 12.8us
Instructions / frame @ 60MHz = (12.8us/frame) / (50ns/instruction)	= 256
instructions/frame	
Instructions / frame @ 66MHz = (12.8us/frame) / (45ns/instruction)	= 284
instructions/frame	
Required instructions / frame	= 250 instructions/frame

FIG. 41

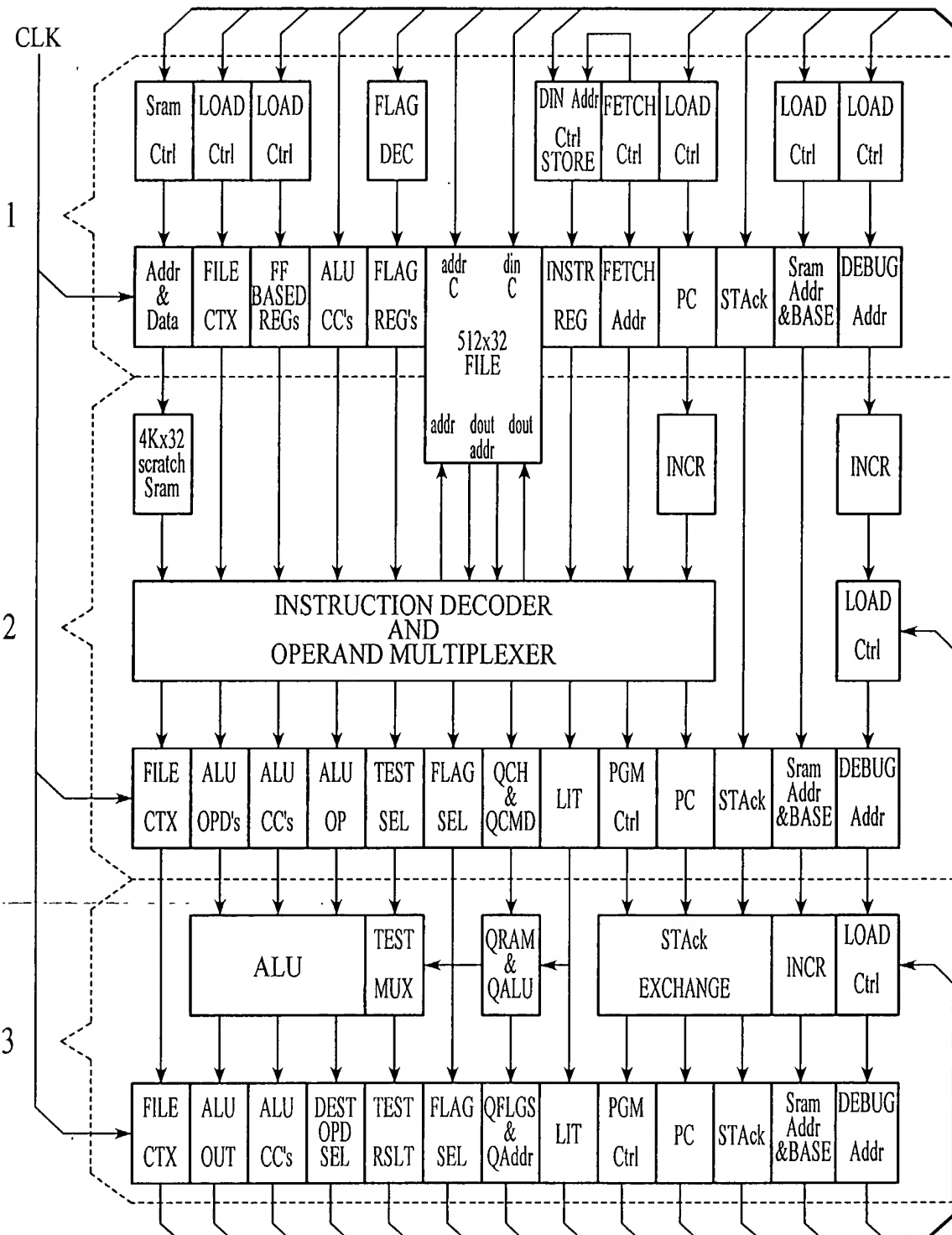


FIG. 42

INSTRUCTION-WORD FORMAT

<u>TYPE</u>	<u>[55:49]</u>	<u>[48:47]</u>	<u>[46:42]</u>	<u>[41:33]</u>	<u>[32:24]</u>	<u>[23:16]</u>	<u>[15:00]</u>
Jcc	0b00000000	0b00,	AluOp,	OpdASel,	OpdBSel,	TstSel, Literal	
Jmp	0b00000000	0b01,	AluOp,	OpdASel,	OpdBSel,	FlgSel, Literal	
Jsr	0b00000000	0b10,	AluOp,	OpdASel,	OpdBSel,	FlgSel, Literal	
Rts	0b00000000	0b11,	AluOp,	OpdASel,	OpdBSel,	0hff,	Literal
Nxt	0b00000000	0b11,	AluOp,	OpdASel,	OpdBSel,	FlgSel, Literal	
Map	MapAddr	0BXX, 0BXXXXXX, 0BXXXXXXXXXX, 0BXXXXXXXXXX, 0HXX, 0HXXXX					

FIG. 43

SEQUENCER BEHAVIOR

```

if (MapEn & (MapAddr != 0b00000000)){           //re-map instr
    Stackc = Stackc;
    StackB = StackB;
    StackA = StackA;
    InstrAddr = 0h8000 | Pc[2:0] | (MapAddr << 3);
    Pc = InstrAddr + (Execute & ~DbgMd);
    Fetch = DbgMd ? DbgAddr:InstrAddr;
    DbgAddr = DbgAddr + (Execute & DbgMd);}

else if (PgmCtrl == Jcc){                         //conditional jump
    Stackc = Stackc;
    StackB = StackB;
    StackA = StackA;
    InstrAddr = ~Tst@TstSel ? Pc:(AluDst==Pc) ? AluOut:Literal;
    Pc = InstrAddr + (Execute & ~DbgMd);
    Fetch = DbgMd ? DbgAddr:InstrAddr;
    DbgAddr = DbgAddr + (Execute & DbgMd);}

else if (PgmCtrl == Jmp){                         //jump
    Stackc = Stackc;
    StackB = StackB;
    StackA = StackA;
    InstrAddr = (AluDst == Pc) ? AluOut:Literal;
    Pc = InstrAddr + (Execute & ~DbgMd);
    Fetch = DbgMd ? DbgAddr:InstrAddr;
    DbgAddr = DbgAddr + (Execute & DbgMd);}

else if (PgmCtrl == JsR){                         //jump subroutine
    Stackc = StackB;
    StackB = StackA;
    StackA = Pc;
    InstrAddr = (AluDst == Pc) ? AluOut:Literal;
    Pc = InstrAddr + (Execute & ~DbgMd);
    Fetch = DbgMd ? DbgAddr:InstrAddr;
    DbgAddr = DbgAddr + (Execute & DbgMd);}

else if (FlgSel == Rts){                         //return subroutine
    InstrAddr = StackA;
    StackA = StackB;
    StackB = Stackc;
    Stackc = ErrVec;
    Pc = InstrAddr + (Execute & ~DbgMd);
    Fetch = DbgMd ? DbgAddr:InstrAddr;
    DbgAddr = DbgAddr + (Execute & DbgMd);}

else {
    InstrAddr = Pc;                               //continue
    StackA = StackA;
    StackB = StackB;
    Stackc = Stackc;
    Pc = InstrAddr + (Execute & ~DbgMd);
    Fetch = DbgMd ? DbgAddr:InstrAddr;
    DbgAddr = DbgAddr + (Execute & DbgMd);}

```

FIG. 44

ALU OPERATIONS

AluOp	OPERATION	
0b00000	$A = (A \& \sim(1 \ll B));$ $C = 0; V = (B \geq 32) ? 1:0;$	//bit clear
0b00001	$A = (A \& B);$ $C = 0; V = 0;$	//logical and
0b00010	$A = (\text{Literal} \& B);$ $C = 0; V = 0;$	//logical and
0b00011	$A = (\sim \text{Literal} \& B);$ $C = 0; V = 0;$	//logical and not
0b00100	$A = (A (1 \ll B));$ $C = 0; V = (B \geq 32) ? 1:0;$	//bit set
0b00101	$A = (A B);$ $C = 0; V = 0;$	//logical or
0b00110	$A = (\text{Literal} B);$ $C = 0; V = 0;$	//logical or
0b00111	$A = (\sim \text{Literal} B);$ $C = 0; V = 0;$	//logical or not
0b01000	for (i=31; i>=0; i--) if B[i] continue; A=i; $C = 0; V = (B) ? 0:1;$	//priority enc
0b01001	$A = (A \wedge B);$ $C = 0; V = 0;$	//logical xor
0b01010	$A = (\{\text{Literal}\} \wedge B);$ $C = 0; V = 0;$	//logical xor
0b01011	$A = (\{\sim \text{Literal}\} \wedge B);$ $C = 0; V = 0;$	//logical xor not
0b01100	$A = B;$ $C = 0; V = 0;$	//move
0b01101	$A = B[31:24] \wedge B[23:16] \wedge B[15:08] \wedge B[07:00];$ $C = 0; V = 0;$	//hash
0b01110	$A = \{B[23:16], B[31:24], B[07:00], B[15:08]\};$ $C = 0; V = 0;$	//swap bytes
0b01111	$A = \{B[15:00], B[31:16]\};$ $C = 0; V = 0;$	//swap doublets

FIG. 45

<u>AluOp</u>	<u>FUNCTION</u>	
0b10000	$A = (A + B);$ $C = (A + B)[32]; V = 0;$	//add B
0b10001	$A = (A + B + C);$ $C = (A + B + C)[32]; V = 0;$	//add B, carry
0b10010	$A = (\text{Literal} + B);$ $C = (\text{Literal} + B)[32]; V = 0;$	//add constant
0b10011	$A = (-\text{Literal} + B);$ $C = (-\text{Literal} + B)[32]; V = 0;$	//sub constant
0b10100	$A = (A - B);$ $C = (A - B)[32]; V = 0;$	//sub B
0b10101	$A = (A - B - \sim C);$ $C = (A - B - \sim C)[32]; V = 0;$	//sub B, borrow
0b10110	$A = (-A + B);$ $C = (-A + B)[32]; V = 0;$	//sub A
0b10111	$A = (-A + B - \sim C);$ $C = (-A + B - \sim C)[32]; V = 0;$	//sub A, borrow
0b11000	$A = (A \ll B);$ $C = A[31]; V = (B \geq 32) ? 0:1;$	//shift left A
0b11001	$A = (B \ll \text{Literal});$ $C = B[31]; V = (\text{Literal} \geq 32) ? 0:1;$	//shift left B
0b11010	$A = (B \ll 1);$ $C = B[31]; V = 0;$	//shift left B
0b11011	$n = (A - B);$ $C = (A - B)[32]; V = 0;$	//compare
0b11100	$A = (A \gg B);$ $C = A[0]; V = (B \geq 32) ? 1:0;$	//shift right A
0b11101	$A = (B \gg \text{Literal});$ $C = A[0]; V = (\text{Literal} \geq 32) ? 1:0;$	//shift right B
0b11110	$A = (B \gg 1);$ $C = A[0]; V = 0;$	//shift right B
0b11111	$n = (B - A);$ $C = (B - A)[32]; V = 0;$	//compare

FIG. 46

OpdSel SELECTED OPERANDs

0b0000aaaaa	File	File@ (OpdSel[4:0] FileBase); Allows paged access to any part of the register file.
0b0001aaaaa	CpuReg	File@ {2'b11, CpuId, OpdSel[4:0]}; Allows direct access to Cpu specific registers.
0b001XXXXXX	reserved	Reserved for future expansion.
0b0100000XX	CpuStatus	0b00000000000000BHD00000000000000CC This is a read-only register providing information about the Cpu executing (OpdSel[1:0]) cycles after the current cycle. "CC" represents a value indicating the Cpu. Currently, only CpuId values of 0, 1 and 2 are returned. "H" represents the current state of Hlt, "D" indicates DbgMd and "B" indicates BigMd. Writing this register has no effect.
0b0100001XX	reserved	Reserved for future expansion.
0b0100010XX	Pc	0x0000AAAA Writing to this address causes the program control logic to use AluOut as the new Pc value in the event of a Jmp, Jcc or Jsrl instruction for the Cpu executing during the current cycle. If the current instruction is Nxt, Map, or Rts, the register write has no effect. Reading this register returns the value in Pc for the Cpu executing (OpdSel[1:0]) cycles after the current cycle.
0b0100011XX	DbgAddr	0xD000AAAA Writing to this register alters the contents of the debug address register (DbgAddr) for the Cpu executing (OpdSel[1:0]) cycles after the current cycle. DbgAddr provides the fetch address for the control-store when DbgMd has been selected and the Cpu is executing. DbgAddr is also used as the control-store address when performing a WrWcs@DbgAddr or RdWcs@DbgAddr operation. "D" represents bit 31 of the register. It is a general purpose flag that is used for event indication during simulation. Reading this register returns a value of 0x00000000.
0b01001XXXX	reserved	Reserved for future expansion.
0b010100000	RamAddr {0b1CCC, 0x000, 0b1, AAAA} RamAddr = AluOut[15] ? AluOut : (AluOut RamBase); PrevCC = AluOut[31] ? CCC : AluCC;	

A read/write register. When reading this register, the Alu condition codes from the previous instruction are returned together with RamAddr.

<u>bit</u>	<u>name</u>	<u>description</u>
31		Always 1.
30	PrevC	Previous Alu Carry.
29	PrevV	Previous Alu Overflow.
28	PrevZ	Previous Alu Zero.
27:16		Always 0.
15		Always 1.
14:0	RamAddr	Contents of last Sram address used.

When writing this register, if alu_out[31] is set, the previous condition codes will be overwritten with bits 30:28 of AluOut. If AluOut[15] is set, bits 14:0 will be written to the RamAddr. If AluOut[15] is not set, bits 14:0 will be ored with the contents of the RamBase and written to the RamAddr

FIG. 47

<u>OpdSel</u>	<u>SELECTED OPERANDs</u>
0b010100001	<p>AddrRegA 0x0000AAAA</p> <p>AddrRegA = AluOut;</p> <p>A read/write operand which loads AddrRegA used to provide the address for read and write operations. When AddrRegA[15] is set, the contents will be presented directly to the ram. When AddrRegA[15] is reset, the contents will first be ored with the contents of the RamBase register before presentation to the ram. Writing to this register takes priority over Literal loads using FlgOp. Reading this register returns the current value of the register.</p>
0b010100010	<p>AddrRegB 0x0000AAAA</p> <p>AddrRegB = AluOut;</p> <p>A read/write operand which loads AddrRegB used to provide the address for read and write operations. When AddrRegB[15] is set, the contents will be presented directly to the ram. When AddrRegB[15] is reset, the contents will first be ored with the contents of the RamBase register before presentation to the ram. Writing to this register takes priority over Literal loads using FlgOp. Reading this register returns the current value of the register.</p>
0b010100011	<p>AddrRegAb 0x0000AAAA</p> <p>AddrRegA = AluOut; AddrRegB = AluOut;</p> <p>A destination only operand which loads AddrRegB and AddrRegA used to provide the address for read and write operations. Writing to this register takes priority over Literal loads using FlgOp. Reading this register returns the value 0x00000000.</p>
0b010100100	<p>RamBase 0x0000AAAA</p> <p>RamBase = AluOut;</p> <p>A read/write register which provides the base address for ram read and write cycles. When RamAddr[15] is set, the contents will not be used. When RamAddr[15] is reset, the contents will first be ored with the contents of the RamBase register before presentation to the ram. Reading this register returns the value for the current Cpu.</p>
0b010100101	<p>FileBase 0b000000000000000000000000AAAAA</p> <p>FileBase = AluOut; FileAddr = OpdSel[8] ? OpdSel:(OpdSel + FileBase);</p> <p>A read/write register which provides the base address for file read and write cycles. When OpdSel[8] is set, the contents will not be used and OpdSel will be presented directly to the address lines of the file. When OpdSel[8] is reset, the contents will first be ored with the contents of the FileBase register before presentation to the file. Reading this register returns the value for the current Cpu.</p>
0b010100110	<p>InstrRegL 0xIIIIIII</p> <p>This is a read-only register which returns the contents of InstrReg[31:0]. Writing to this register has no effect.</p>
0b010100111	<p>InstrRegH 0x00IIIII</p> <p>This is a read-only register which returns the contents of InstrReg[55:32]. Writing to this register has no effect.</p>

FIG. 48

<u>OpdSel</u>	<u>SELECTED OPERANDs</u>	
0b010101000	Minus1	0xffffffff This is a read-only register which supplies a value 0xffffffff.. Writing to this register has no effect.
0b010101001	FreeTime	A free-running timer with a resolution of 1.00 microseconds and a maximum count of 71 minutes. This timer is cleared during reset.
0b010101010	LiteralL	Instr[15:0] A read-only register. Writing to this register has no effect
0b010101011	LiteralH	Instr[15:0]<<16; A read-only register. Writing to this register has no effect
0b010101100	MacData - Writing to this address loads the AluOut data into the MacData register for use during Mac operations. The Mac operation, resulting from writing to the MacOp register, determines the definition of the MacData register contents as follows.	
	<u>MacOp</u>	<u>MacData definition</u>
	Mstop	0bXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX MacData is not used for the StopM operation.
	WrMcfg	hrstl , rsvd , rsvd , ercen , fulld , hrstl , hugen , nopre , paden , prtyl , xdll0 , ipgr1[6:0] , ipgr2[6:0] , ipgt[6:0] . Loads the MacCfg register with the contents of the MacData register. Refer to LSI Logic's <i>Ethernet-110 Core Technical Manual</i> for detailed definitions of these bits.
	WrMrng	0bXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXSSSSSSSSSS Loads seed[10:0] into the Mac's random number generator.
	RdPhy	0bXXXXRRRRXXXXPPPPXXXXXXXXXXXXXXXXXXXX Reads register[R] of phy[P].
	WrPhy	0bXXXXRRRRXXXXPPPPDDDDDDDDDDDDDDDDDDDD Writes register[R] of phy[P] with MacData[15:0] .
Reading this register returns prsd[15:0] of Mac0 which contains phy status data returned to the Mac at the completion of a RdPhy command. This data is invalid while MacBsy is asserted as a result of a RdPhy command. Refer to the appropriate phy technical manual for a definition of the phy register contents.		

FIG. 49

FIG. 50A

FIG. 50B

FIG. 50C

FIG. 50

<u>OpdSel</u>	<u>SELECTED OPERANDs</u>
---------------	--------------------------

0b010101101 **MacOp** - A write only register. Writing to this address loads the **MacSel** register and starts execution of the specified operation as follows.

<u>AluOut</u>	<u>description</u>
0xXXXXXX0XM	Mstop - Halts execution of a MacOp for Mac[M] . The user must wait for MacBsy to be deasserted before issuing another command or changing the contents of MacData .
0xXXXXXX1XM	WrMcfg - Writes the contents of MacData to the MacCfg register of Mac[M] . The user must wait for MacBsy to be deasserted before issuing another command or changing the contents of MacData .
0xXXXXXX2XM	WrMrng - Writes the contents of MacData to the seed register of Mac[M] . The user must wait for MacBsy to be deasserted before issuing another command or changing the contents of MacData .
0xXXXXXX3XM	RdPhy - Reads the contents of reg[R] for phy[P] on the MII management bus of Mac[M] . The contents may be read from MacData after MacBsy has been deasserted.
0xXXXXXX4XM	WrPhy - Writes the contents of MacData [15:0] to e reg[R] of phy[P] on the MII management bus of Mac[M] . The user must wait for MacBsy to be deasserted before issuing another command or changing the contents of MacData .
0xXXXXXX8XM	WrAddrAL - Writes the contents of MacData [15:0] to MacAddrA [15:0] for Mac[M] .
0xXXXXXX9XM	WrAddrAH - Writes the contents of MacData [11:0] to MacAddrA [47:16] for Mac[M] .
0xXXXXXXaXM	WrAddrBL - Writes the contents of MacData [15:0] to MacAddrB [15:0] for Mac[M] .
0xXXXXXXbXM	WrAddrBH - Writes the contents of MacData [11:0] to MacAddrB [47:16] for Mac[M] .

b010101110 **ChCmd** A write-only register.

<u>bit</u>	<u>name</u>	<u>description</u>
31:11	reserved	Data written to these bits is ignored.
10:8	command	0 - Stops execution of the current operation and clears the corresponding event flag. 1 - Transfer data from ExtMem to ExtMem. 2 - Transfer data from Pci to ExtMem. 3 - Transfer data from ExtMem to Pci. 4 - Transfer data from Sram to ExtMem. 5 - Transfer data from ExtMem to Sram. 6 - Transfer data from Pci to Sram. 7 - Transfer data from Sram to Pci.
07:05	reserved	Data written to these bits is ignored.
04:00	ChId	Provides the channel number for the channel command.

FIG. 50A

0b010101110	ChEvt	A read-only register.	
<u>bit</u>	<u>name</u>	<u>description</u>	
31:00	ChDn	Each bit represents the done flag for the respective dma channel. These bits are set by a dma sequencer upon completion of the channel command. Cleared when the processor writes 0 to the corresponding ChCmd register.	
0b010101111	GenEvt	A read-only register.	
<u>bit</u>	<u>name</u>	<u>description</u>	
31	PciRdEvt	Indicates that a PCI initiator is attempting to read a mproc. register.	
30	PciWrEvt	Indicates that a PCI initiator has posted a write to a mproc. register.	
29	TimeEvt	An event which occurs once every 2.00 milliseconds.	
28:00	reserved	Reserved for future use.	
0b010110000	QCtrl	A write-only register used to select and manipulate a Q.	
<u>bit</u>	<u>name</u>	<u>description</u>	
31:11	reserved	Data written to these bits are ignored.	
10:8	QSz	Used only during InitQ operations to specify the size of the QBdy in Dram. 7 – Queue depth is 32K entries (128KB). 6 – Queue depth is 16K entries (64KB). 5 – Queue depth is 8K entries (32KB). 4 – Queue depth is 4K entries (16KB). 3 – Queue depth is 2K entries (8KB). 2 – Queue depth is 1K entries (4KB). 1 – Queue depth is 512 entries (2KB). 0 – Queue depth is 256 entries (1KB).	
7:5	QOp	Specifies the queue operation to perform. 7 – DbIQ Disables all queues. 6 – EnQ Enables all queues. 5 – RdBdy Increments the QBdyRdPtr and increments the QTIWrPtr . 4 – WrBdy Decrements the QBdyWrPtr and increments the QHdRdPtr . 3 – RdQ Returns a queue entry in register QData . 2 – rsvd Reserved. Not to be used. 1 – InitQ Set the queue status to empty and initializes QSz . 0 – ScIQ Selects the QId to be utilized during writes to QData .	

FIG. 50B

4:0	QId	Specifies the queue on which to perform all operations except DbIQ or EnQ .																								
0b010110001	QData	A read/write register. Writing this register will result in the data being pushed on to the selected queue. Reading this register fetches queue data popped off during the previous RdQ operation.																								
0b010110010	reserved	Reserved for future expansion.																								
0b010110011	XcvCtrl	A write-only register used to enable and disable Mac transmit and receive sub-channels.																								
<table> <tr> <th>bit</th><th>name</th><th>description</th></tr> <tr> <td>31:09</td><td>reserved</td><td>Data written to these bits are ignored.</td></tr> <tr> <td>8</td><td>enable</td><td>When set, indicates to the Mac transmit or receive sequencer that the subchannel contains a transmit or receive descriptor.</td></tr> <tr> <td>07:05</td><td>reserved</td><td>Data written to these bits is ignored.</td></tr> <tr> <td>04</td><td>RcvCh</td><td>Selects a Mac receive subchannel when set. Selects a Mac transmit subchannel when cleared.</td></tr> <tr> <td>03</td><td>reserved</td><td>Data written to this bit are ignored.</td></tr> <tr> <td>02</td><td>SubCh</td><td>Selects subchannel B when set or A when reset.</td></tr> <tr> <td>01:00</td><td>MacId</td><td>Provides the Mac number for the subchannel enable bit.</td></tr> </table>			bit	name	description	31:09	reserved	Data written to these bits are ignored.	8	enable	When set, indicates to the Mac transmit or receive sequencer that the subchannel contains a transmit or receive descriptor.	07:05	reserved	Data written to these bits is ignored.	04	RcvCh	Selects a Mac receive subchannel when set. Selects a Mac transmit subchannel when cleared.	03	reserved	Data written to this bit are ignored.	02	SubCh	Selects subchannel B when set or A when reset.	01:00	MacId	Provides the Mac number for the subchannel enable bit.
bit	name	description																								
31:09	reserved	Data written to these bits are ignored.																								
8	enable	When set, indicates to the Mac transmit or receive sequencer that the subchannel contains a transmit or receive descriptor.																								
07:05	reserved	Data written to these bits is ignored.																								
04	RcvCh	Selects a Mac receive subchannel when set. Selects a Mac transmit subchannel when cleared.																								
03	reserved	Data written to this bit are ignored.																								
02	SubCh	Selects subchannel B when set or A when reset.																								
01:00	MacId	Provides the Mac number for the subchannel enable bit.																								
0b010110100	Lru	0x0000000A A read/write operand indicating which of the 16 entries is least recently used. When Reading This register the least recently used entry is returned, after which it is automatically made the most recently used entry. This register should only be read in conjunction with a 'Move' operation of the ALU, else the results are unpredictable. Writing to this register forces the addressed entry to become the least recently used entry.																								
0b010110101	Mru	0x0000000A A write only operand forcing the addressed entry to become the most recently used entry.																								
0b010111000	QInRdy	A read-only register comprising QHd not full flags for each of the 32 queues.																								
0b010111001	QOutRdy	A read-only register comprising QTI not empty flags for each of the 32 queues.																								
0b010111010	QEmpty	A read-only register comprising QEmpty flags for each of the 32 queues.																								
0b010111011	QFull	A read-only register comprising QFull flags for each of the 32 queues.																								
0b0101111XX	reserved	Reserved for future expansion.																								
0b0110XXXXX	Constants	{0b000, OpdSel [4:0]}																								
0b01110XXXX	reserved	Reserved for future expansion.																								

FIG. 50C

OpdSel SELECTED OPERANDs0b01111XXXX Sram OPERATIONS

<u>OpdSel[3]</u>	<u>PostAddrOp</u>
0	nop
1	RamAddr = RamAddr + (OpdSel[1:0]);
<u>OpdSel[2]</u>	<u>transpose_Ctrl</u>
0	don't transpose
1	transpose bytes
<u>OpdSel[1:0]</u>	<u>RamOpdSz</u>
0	quadlet
1	triplet
2	doublet
3	byte

RAM READ ATTRIBUTESSOURCE OPERAND

<u>endian mode</u>	<u>trans- pose</u>	<u>byte offs</u>	<u>Sram data</u>	<u>sz=Q</u>	<u>sz=T</u>	<u>sz=D</u>	<u>sz=B</u>
little	0	0	abcd	abcd	0bcd	00cd	000d
little	0	1	abcX	trap	0abc	00bc	000c
little	0	2	abXX	trap	trap	00ab	000b
little	0	3	aXXX	trap	trap	trap	000a
little	1	0	abcd	dcba	0dcb	00dc	000d
little	1	1	abcX	trap	0cba	00cb	000c
little	1	2	abXX	trap	trap	00ba	000b
little	1	3	aXXX	trap	trap	trap	000a
BIG	0	0	abcd	abcd	0abc	00ab	000a
BIG	0	1	Xbcd	trap	0bcd	00bc	000b
BIG	0	2	XXcd	trap	trap	00cd	000c
BIG	0	3	XXXd	trap	trap	trap	000d
BIG	1	0	abcd	dcba	0cba	00ba	000a
BIG	1	1	Xbcd	trap	0dcb	00cb	000b
BIG	1	2	XXcd	trap	trap	00dc	000c
BIG	1	3	XXXd	trap	trap	trap	000d

RAM WRITE ATTRIBUTESSOURCE OPERAND

<u>endian mode</u>	<u>trans- pose</u>	<u>Opd size</u>	<u>Alu out</u>	<u>OF=0</u>	<u>OF=1</u>	<u>OF=2</u>	<u>OF=3</u>
little	0	Q	abcd	abcd	trap	trap	trap
little	0	T	Xbcd	-bcd	bcd-	trap	trap
little	0	D	XXcd	--cd	-cd-	cd--	trap
little	0	B	XXXd	---d	--d-	-d--	d---
little	1	Q	abcd	dcba	trap	trap	trap
little	1	T	Xbcd	-dcb	dc-	trap	trap
little	1	D	XXcd	--dc	-dc-	dc--	trap
little	1	B	XXXd	---d	--d-	-d--	d---
big	0	Q	abcd	abcd	trap	trap	trap
big	0	T	Xbcd	bcd-	-bcd	trap	trap
big	0	D	XXcd	cd--	-cd-	--cd	trap
big	0	B	XXXd	d---	-d--	--d-	---d
big	1	Q	abcd	dcba	trap	trap	trap
big	1	T	Xbcd	dc-	-dcb	trap	trap
big	1	D	XXcd	dc--	-dc-	--dc	trap
big	1	B	XXXd	d---	-d--	--d-	---d

0b1aaaaaaaa **File** **File@OpdSel[8:0];**
 Allows direct, non-paged, access to the top half of the register file.

FIG. 51

<u>TstSel</u>	<u>SELECTED TEST</u>	
0bX00XXXXX	$Tst = TstSel[7] \wedge AluOut[TstSel[4:0]]$	//Alu bit
0bX0100000	$Tst = TstSel[7] \wedge C$	//carry
0bX0100001	$Tst = TstSel[7] \wedge V$	//error
0bX0100010	$Tst = TstSel[7] \wedge Z$	//zero
0bX0100011	$Tst = TstSel[7] \wedge (Z \mid \sim C)$	//less or equal
0bX0100100	$Tst = TstSel[7] \wedge PrevC$	//previous carry
0bX0100101	$Tst = TstSel[7] \wedge PrevV$	//previous error
0bX0100110	$Tst = TstSel[7] \wedge PrevZ$	//previous zero
0bX0100111	$Tst = TstSel[7] \wedge (PrevZ \ \& \ Z)$	//64b zero
0bX0101000	$Tst = TstSel[7] \wedge QOpDn$	//queue op okay
0bX0101001	$Tst = \text{reserved}$	
0bX010101X	$Tst = \text{reserved}$	
0bX01011XX	$Tst = \text{reserved}$	
0bX0110XXX	$Tst = TstSel[7] \wedge Lock[TstSel[2:0]]$ $Lock(TstSel[2:0]) = 1;$	//tests the current value of //the Lock then set it.
0bX0111XXX	$Tst = TstSel[7] \wedge Lock[TstSel[2:0]]$	//tests the value of Lock.
0bX01XXXXXX	$Tst = \text{reserved}$	
0bX1XXXXXXX	$Tst = \text{reserved}$	

FIG. 52

<u>FlgSel</u>	<u>FLAG OPERATION</u>	
0b00000000	No operation.	
0b00000001	SelfRst	Forces a self reset for the entire chip excluding the PCI configuration registers
0b00000010	SelBigEnd	Selects big-endian mode for ram accesses for the current Cpu.
0b00000011	SelLitEnd	Selects little-endian mode for ram accesses for the current Cpu.
0b00000100	DbIMap	Disable instruction re-mapping for the current Cpu.
0b00000101	EnbMap	Enable instruction re-mapping for the current Cpu.
0b0000011X	reserved	
0b00001XXX	reserved	
0b00010XXX	ClrLck	Lock[FlgSel[2:0]] = 0; Clears the semaphore register bit for the current Cpu only.
0b00011XXX	reserved	
0b0010XXXX	AddrOp	
	<u>FlgSel[3:2]</u>	<u>AddrSelect</u>
	0	RamAddr = Literal[15] ? Literal : (Literal RamBase);
	1	RamAddr = AddrRegA[15] ? AddrRegA : (AddrRegA RamBase);
	2	RamAddr = AddrRegB[15] ? AddrRegB : (AddrRegB RamBase);
	3	if (OpdA == RamAddr) RamAddr = AluOut[15] ? AluOut : (AluOut RamBase); else if (OpdA == ram) RamAddr = AddrRegB[15] ? AddrRegB : (AddrRegB RamBase); else RamAddr = AddrRegA[15] ? AddrRegA : (AddrRegA RamBase);
	<u>FlgSel[1:0]</u>	<u>addr reg load</u>
	0	nop
	1	AddrRegA = Literal;
	2	AddrRegB = Literal;
	3	AddrRegA = Literal; AddrRegB = Literal;
note: When specifying the same register for both the load and select fields, the current value of the register, before it is loaded with the new value, will be used for the ram address.		
0b0011XXXX	reserved	
0b01000000	WrWcsL@Dbg	Causes the bits [31:0] of the control-store at address DbgAddr to be written with the current AluOut data.
0b01000001	WrWcsH@Dbg	Causes the bits [63:32] of the control-store at address DbgAddr to be written with the current AluOut data then increments DbgAddr .
0b01000010	RdWcsL@Dbg	Causes the bits [31:0] of the control-store at address DbgAddr to be moved to file address 0x1ff
0b01000011	RdWcsH@Dbg	Causes the bits [63:32] of the control-store at address DbgAddr to be moved to file address 0x1ff then increments DbgAddr .
0b01000100	reserved	
0b010001XX	Step	Allows the Cpu (FlgSel[1:0]) cycles after the current cycle to execute a single instruction. There is no effect if the Cpu is not halted. An offset of 0 is not allowed.
0b010010XX	PcMd	Selects the Pc as the address source for the control-store during instruction fetches for the Cpu (FlgSel[1:0]) cycles after the current cycle.
0b010011XX	DbgMd	Selects the DbgAddr address register as the address source for the control-store during instruction fetches for the Cpu (FlgSel[1:0]) cycles after the current cycle.
0b010100XX	Hlt	Halts the Cpu (FlgSel[1:0]) cycles after the current cycle.
0b010101XX	Run	Clears Halt for the Cpu (FlgSel[1:0]) cycles after the current cycle.
0b01011XXX	reserved	
0b011XXXXX	reserved	
0b1XXXXXXX	reserved	

FIG. 53

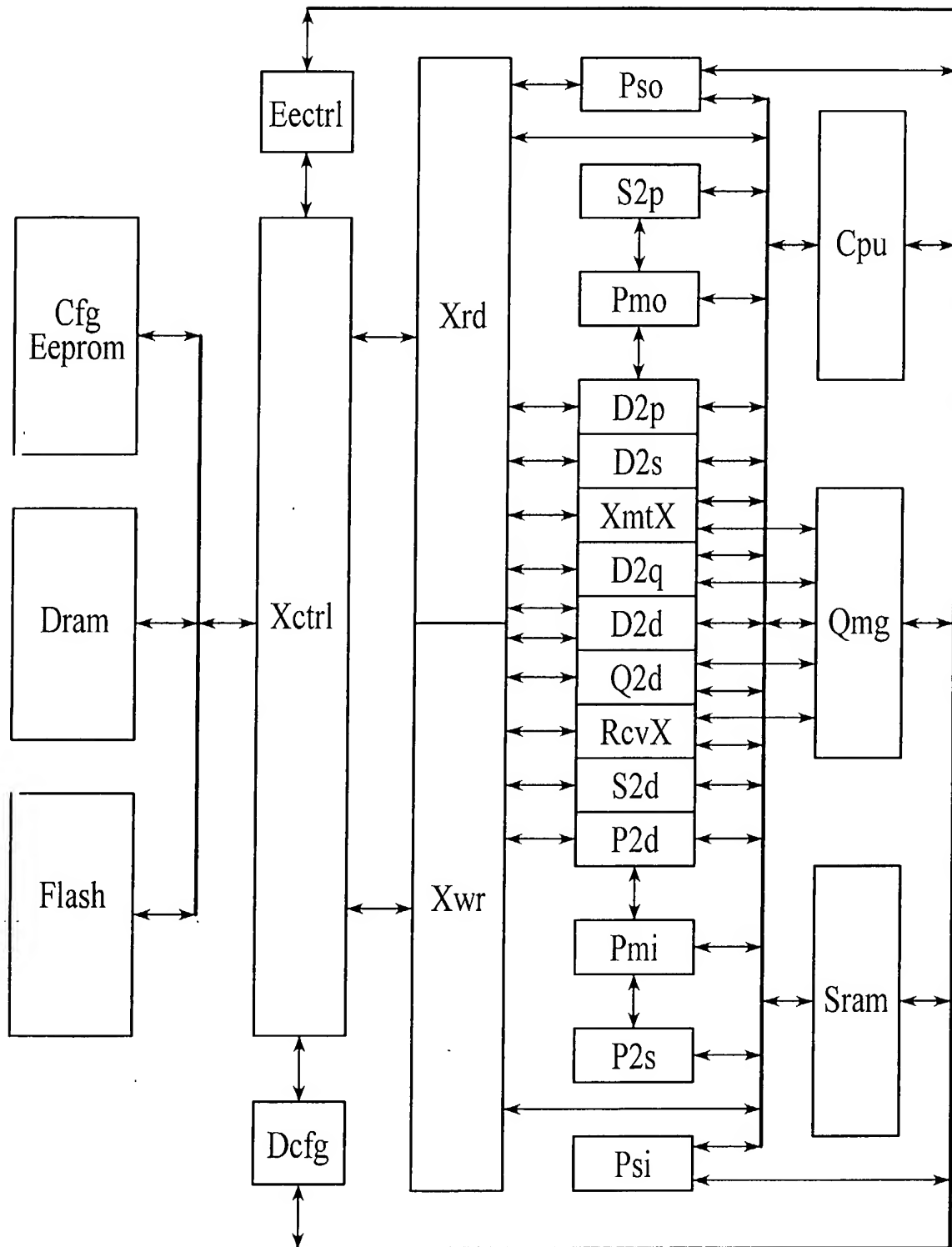


FIG. 54

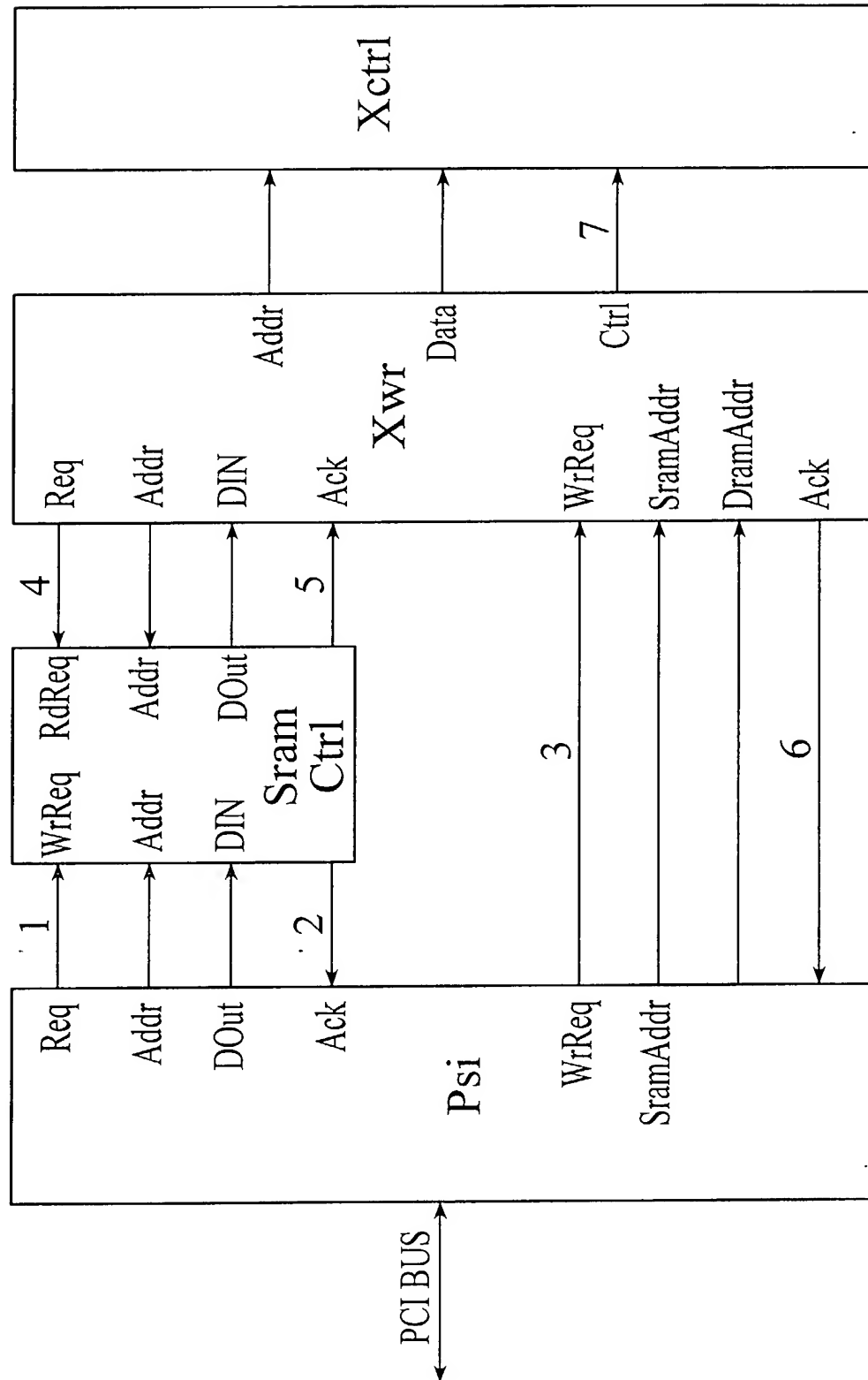


FIG. 55

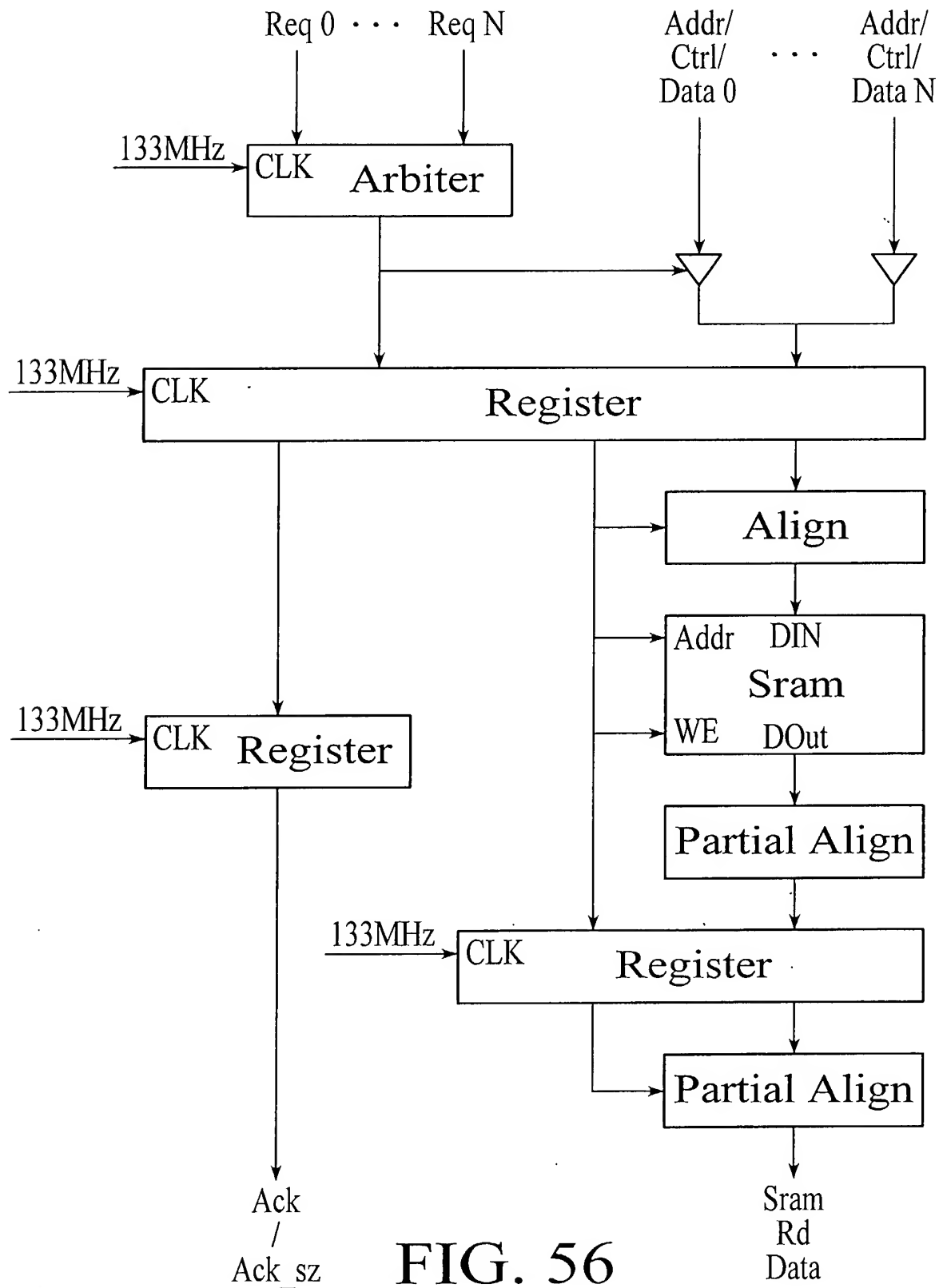


FIG. 56

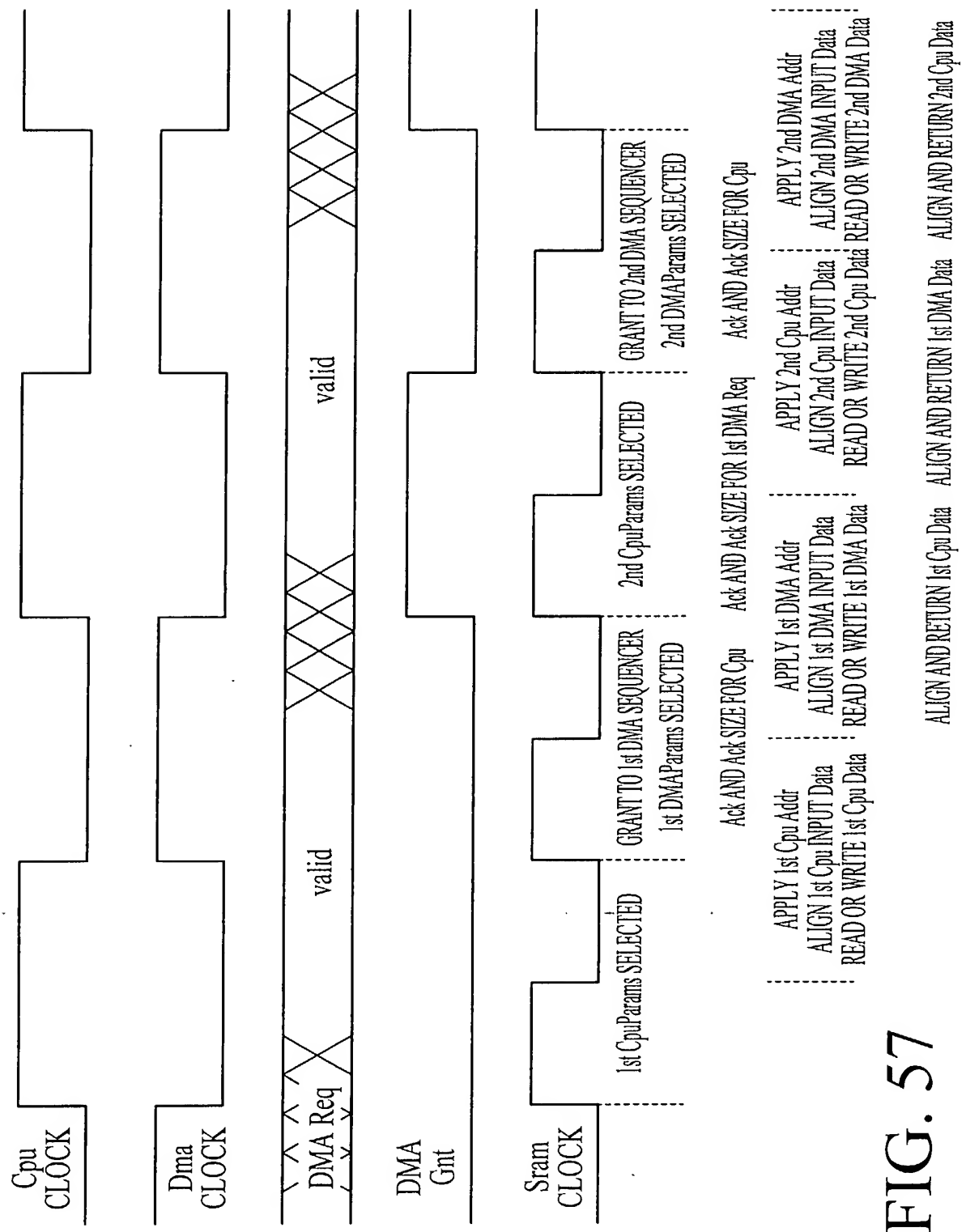


FIG. 57

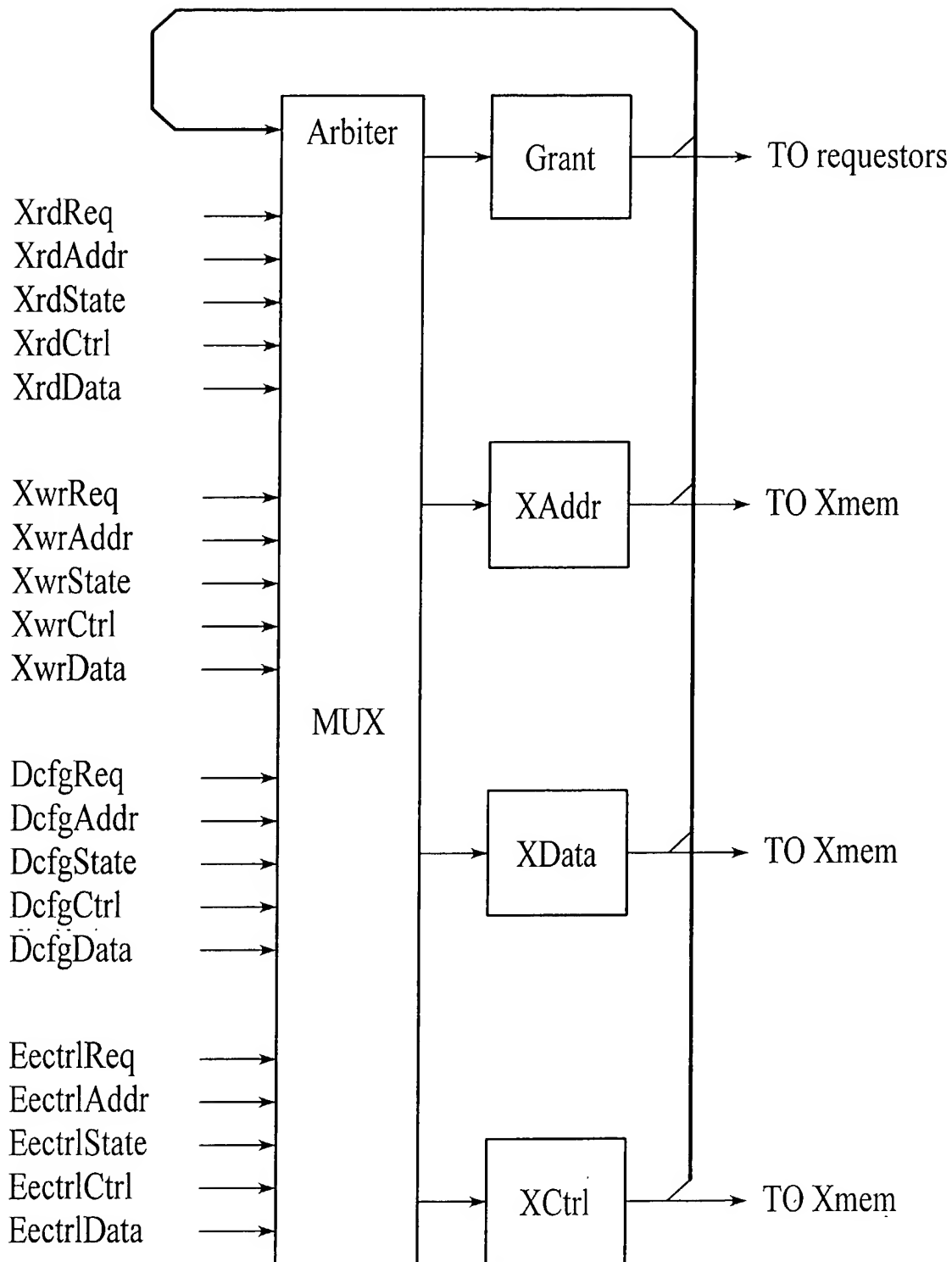


FIG. 58

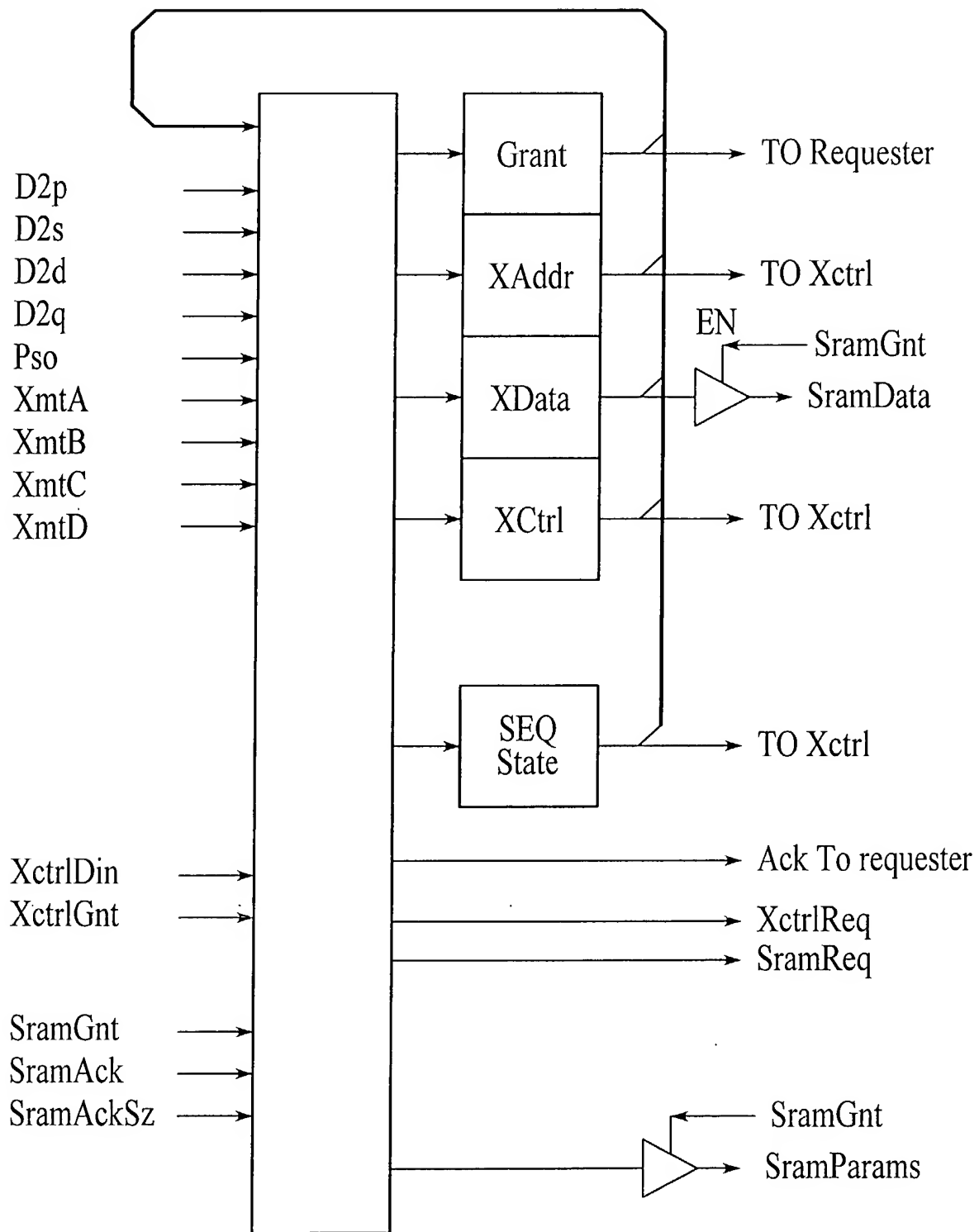


FIG. 59

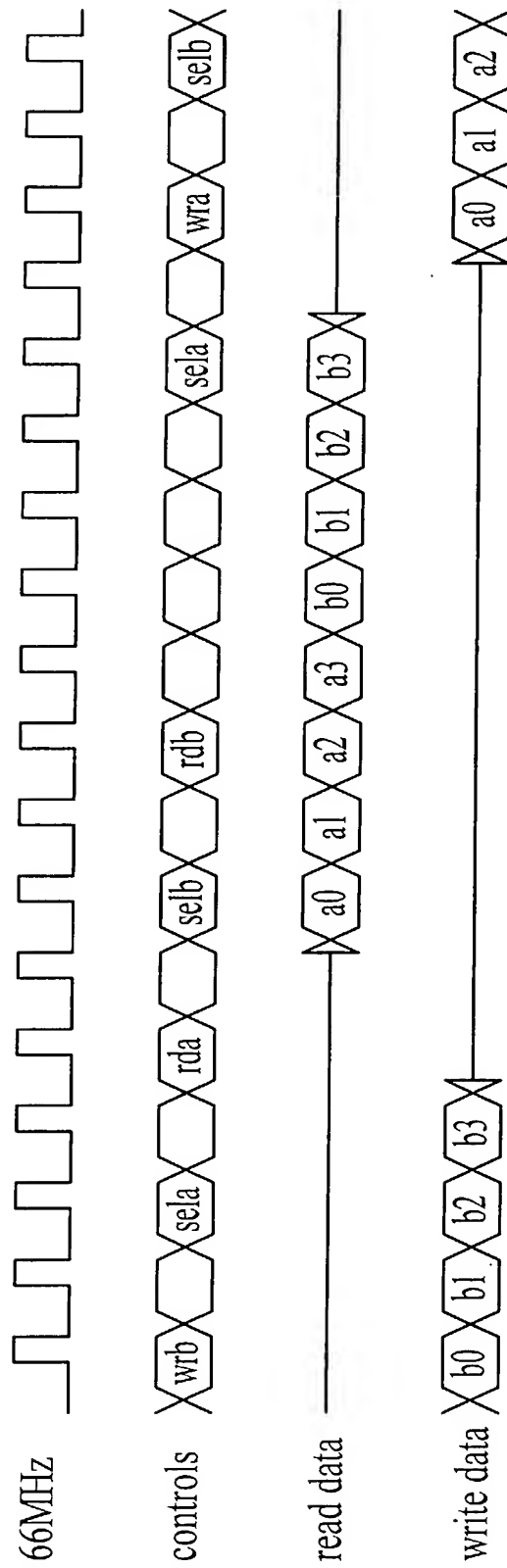


FIG. 60

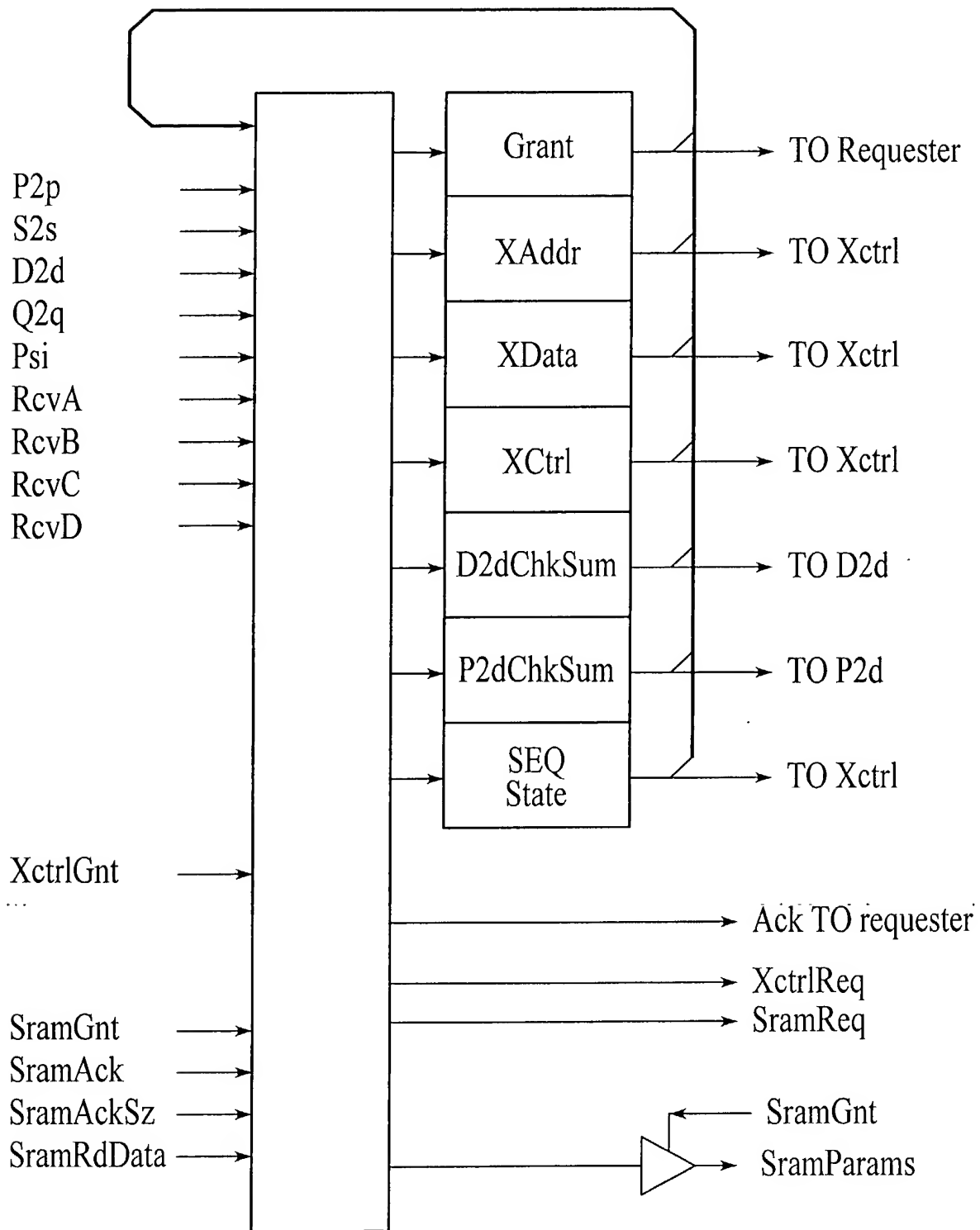


FIG. 61

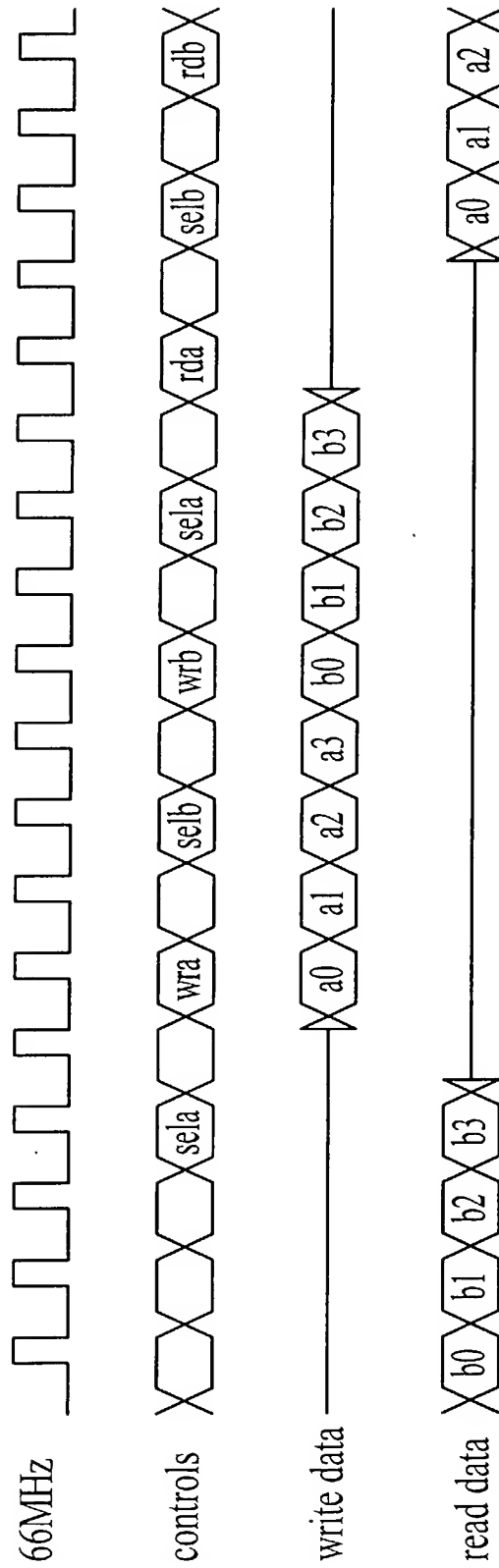


FIG. 62

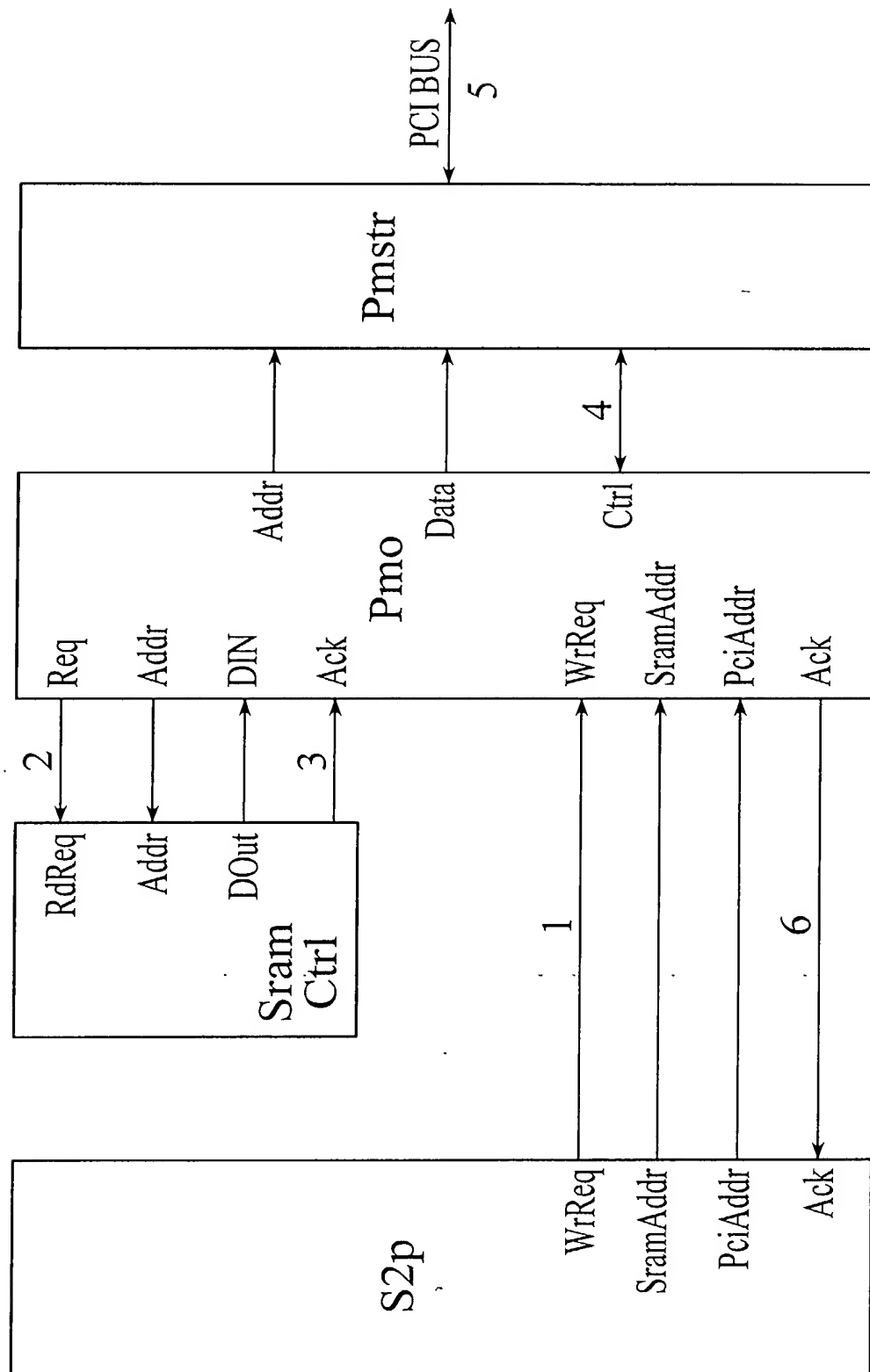


FIG. 63

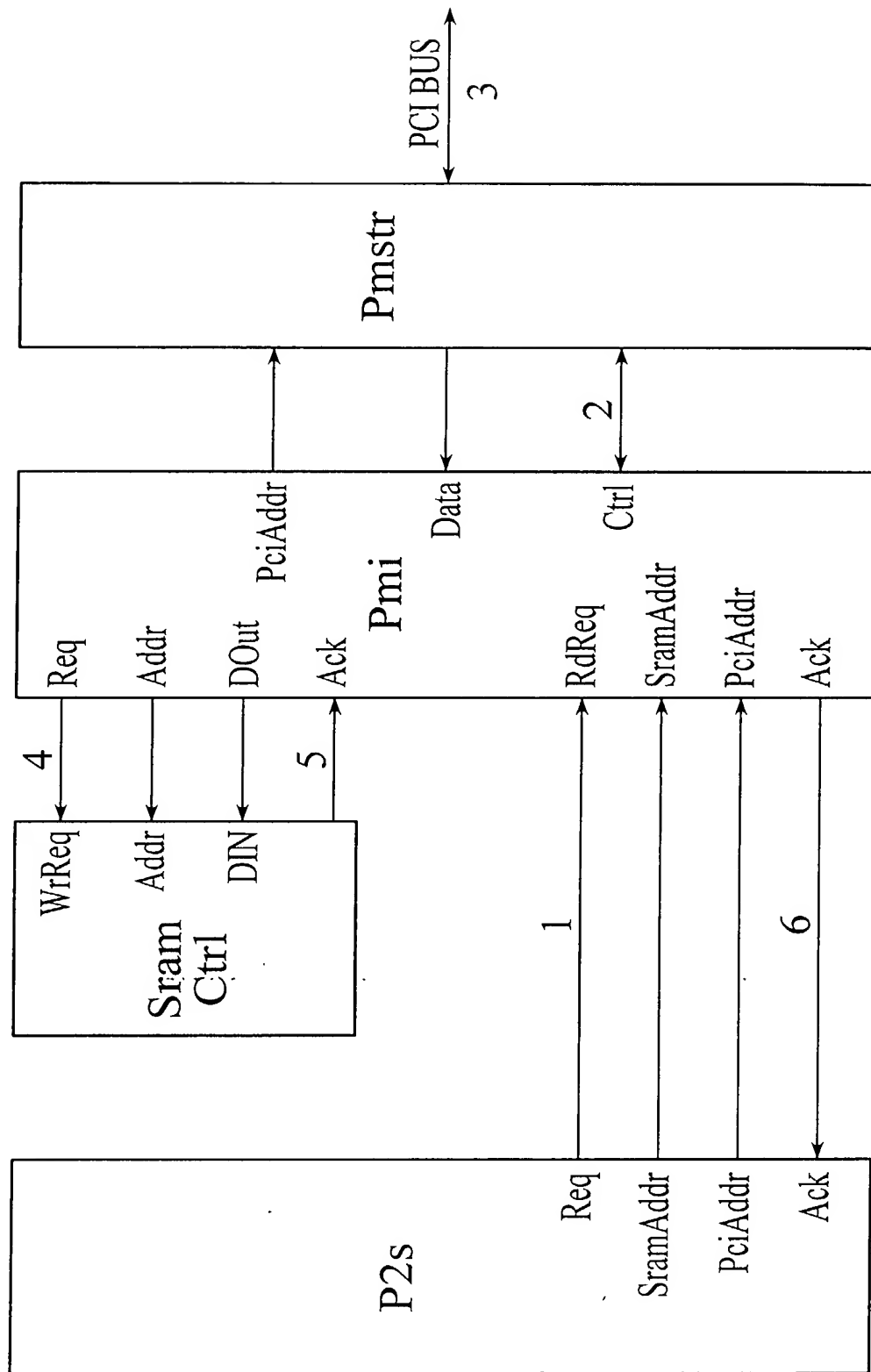


FIG. 64

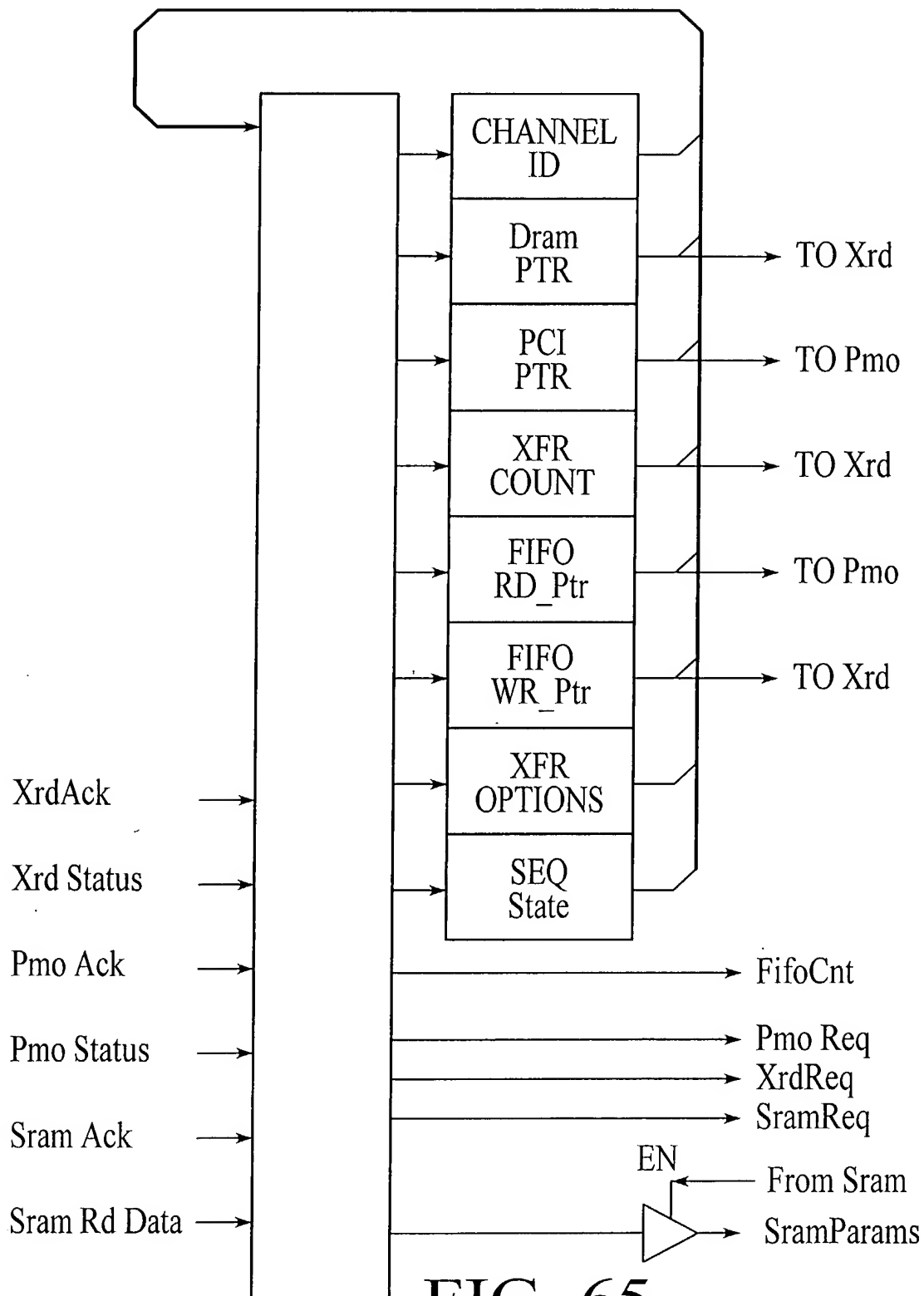


FIG. 65

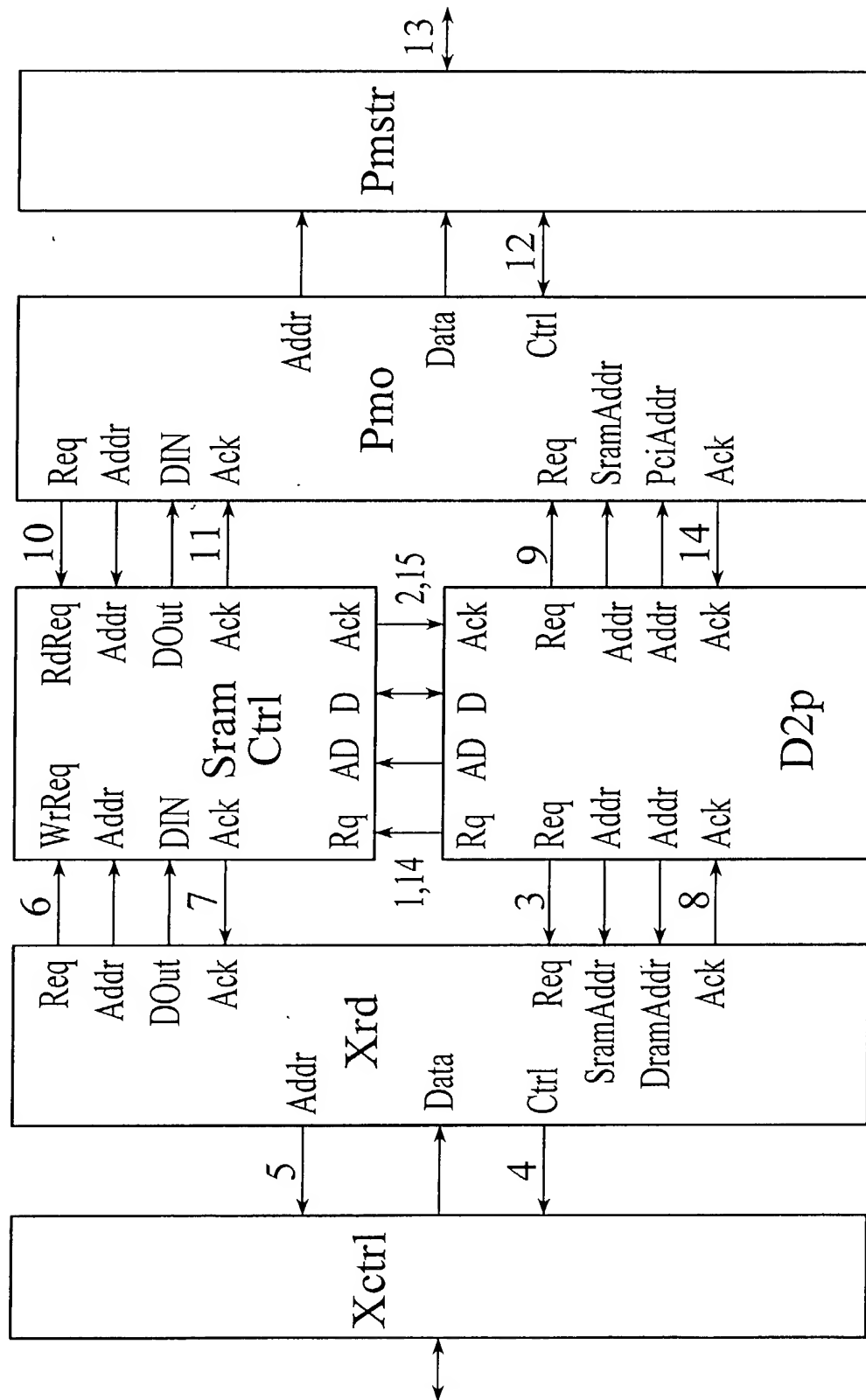
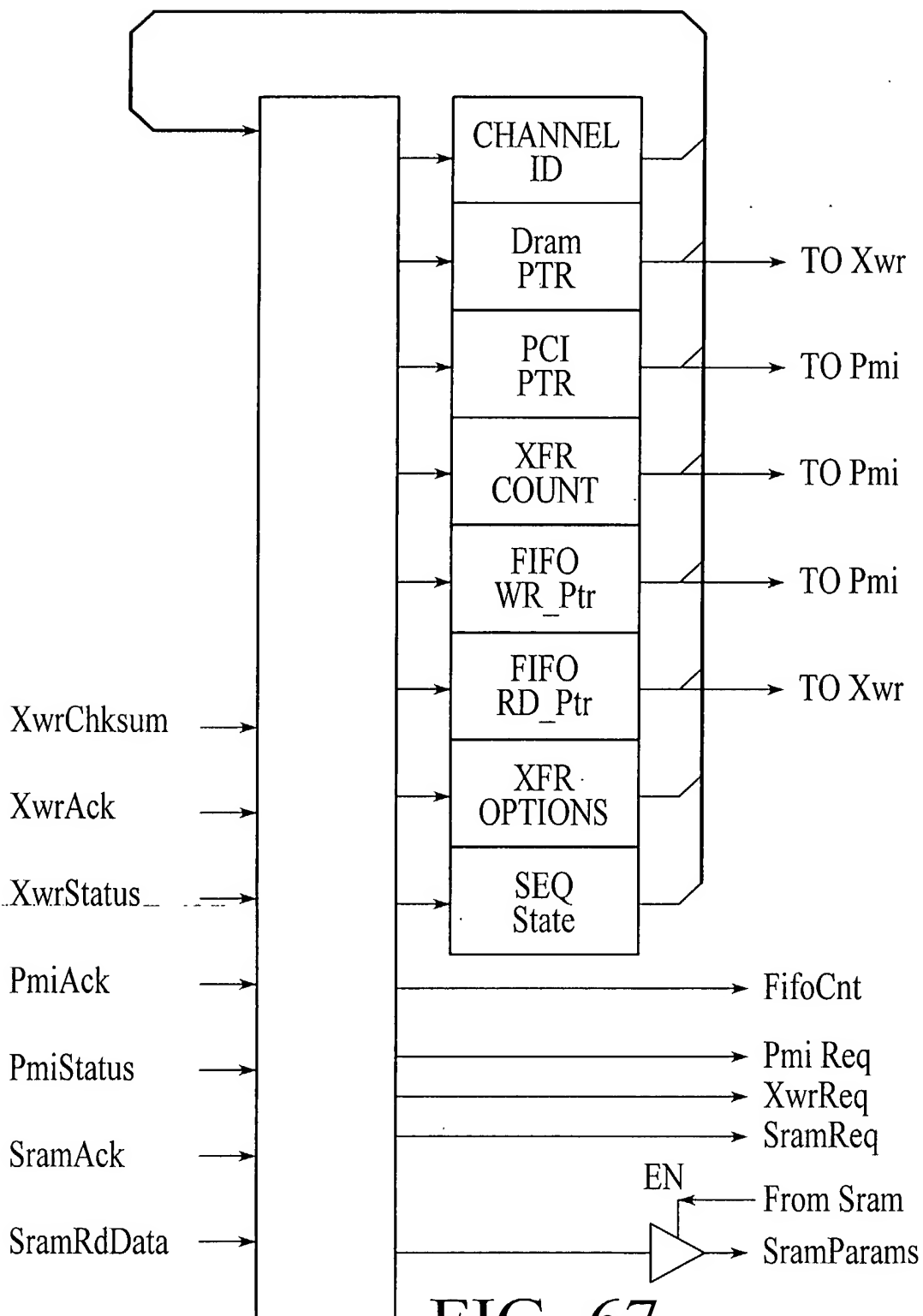


FIG. 66



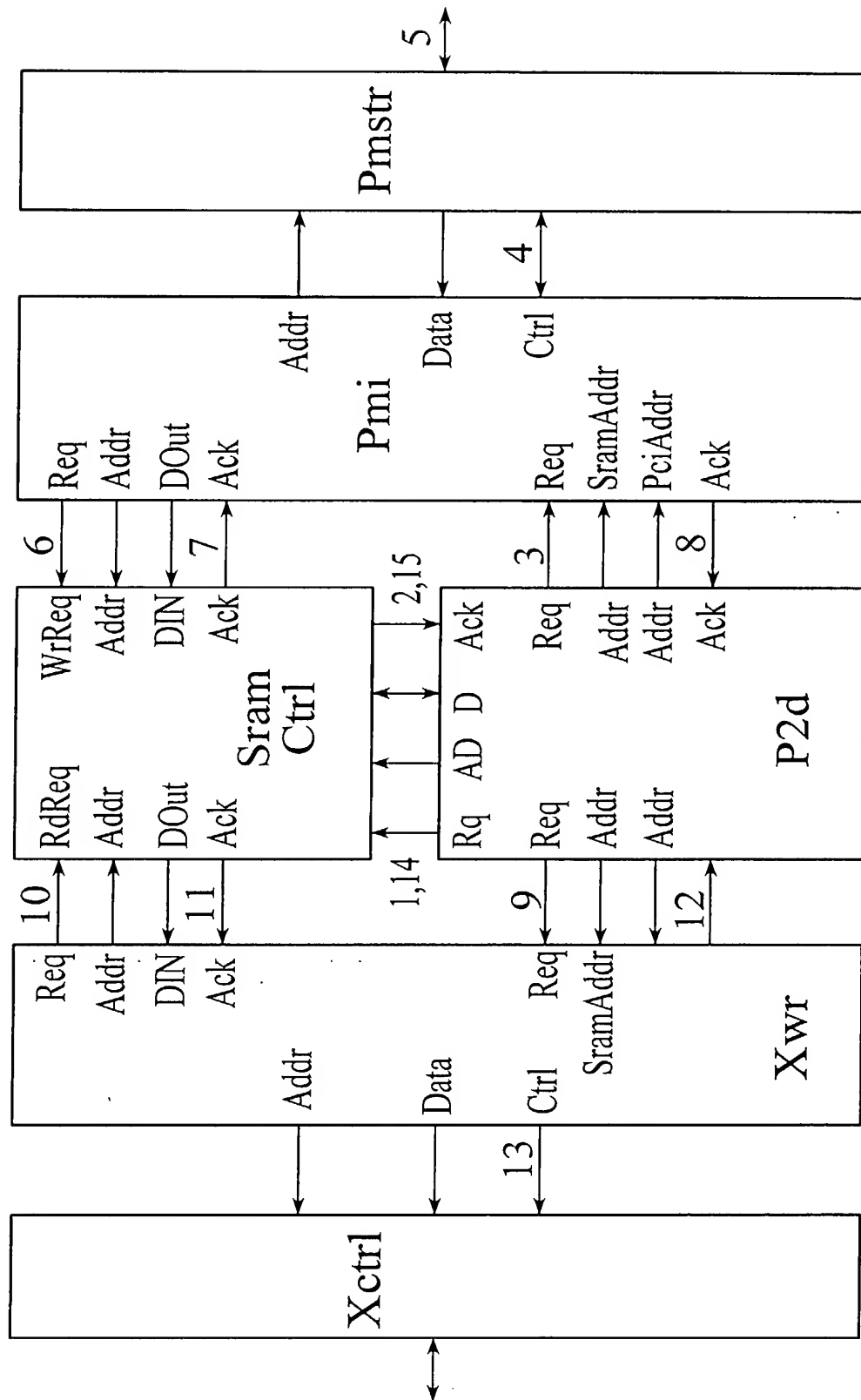


FIG. 68

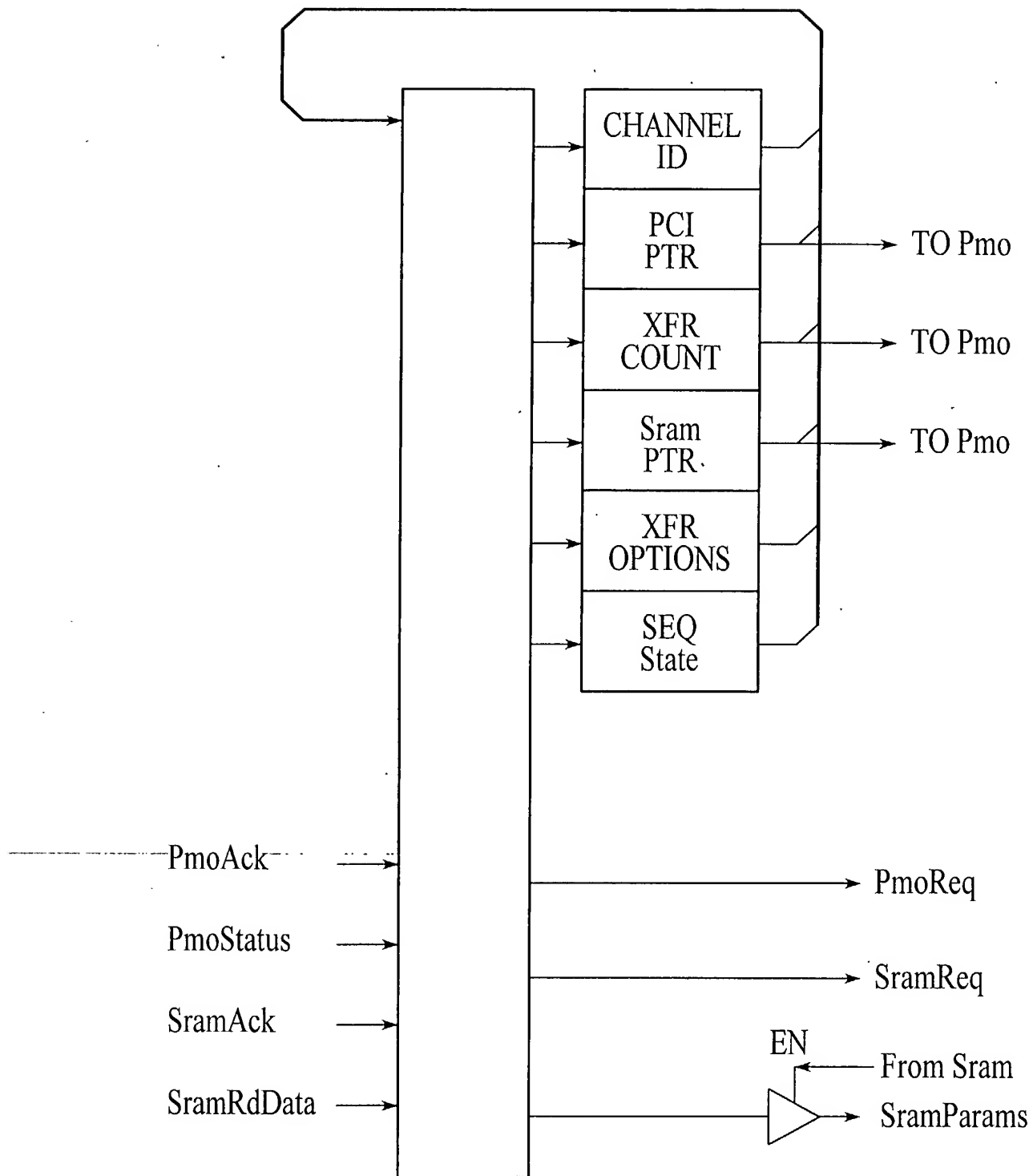


FIG. 69

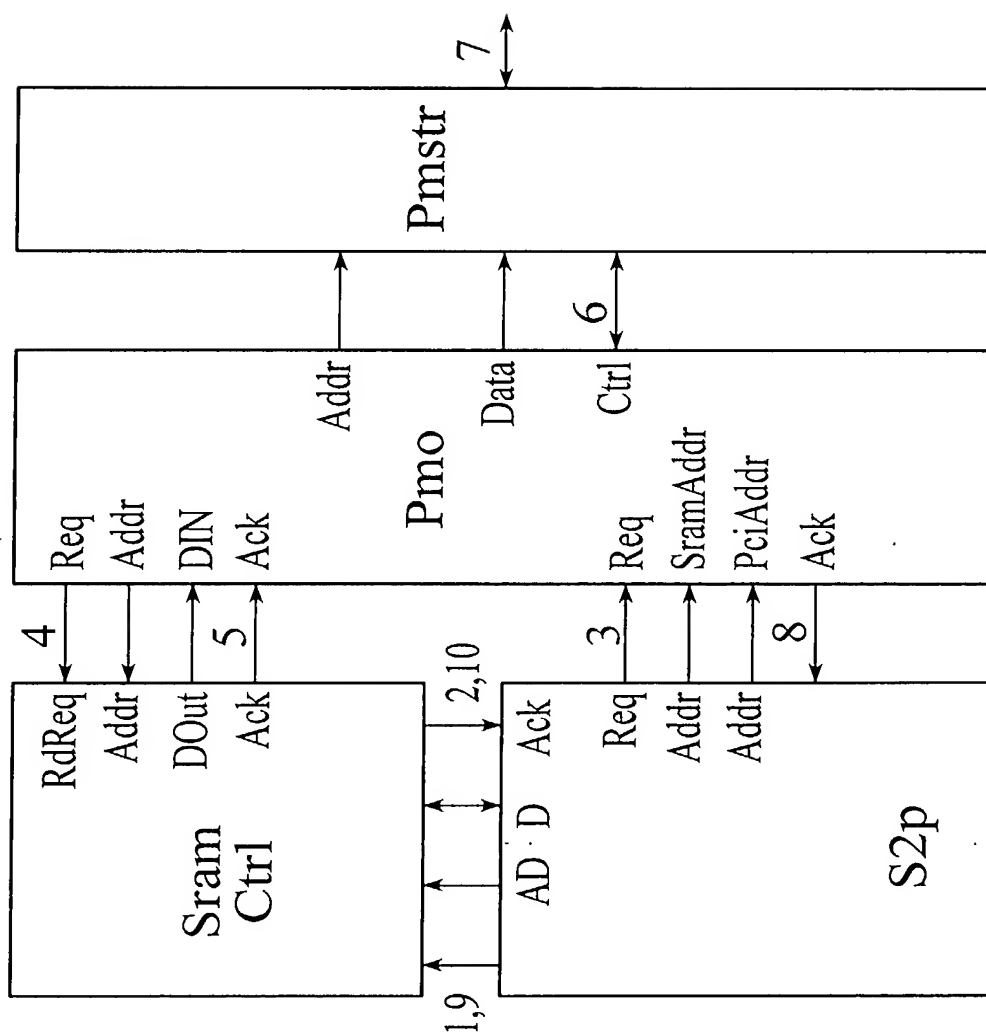


FIG. 70

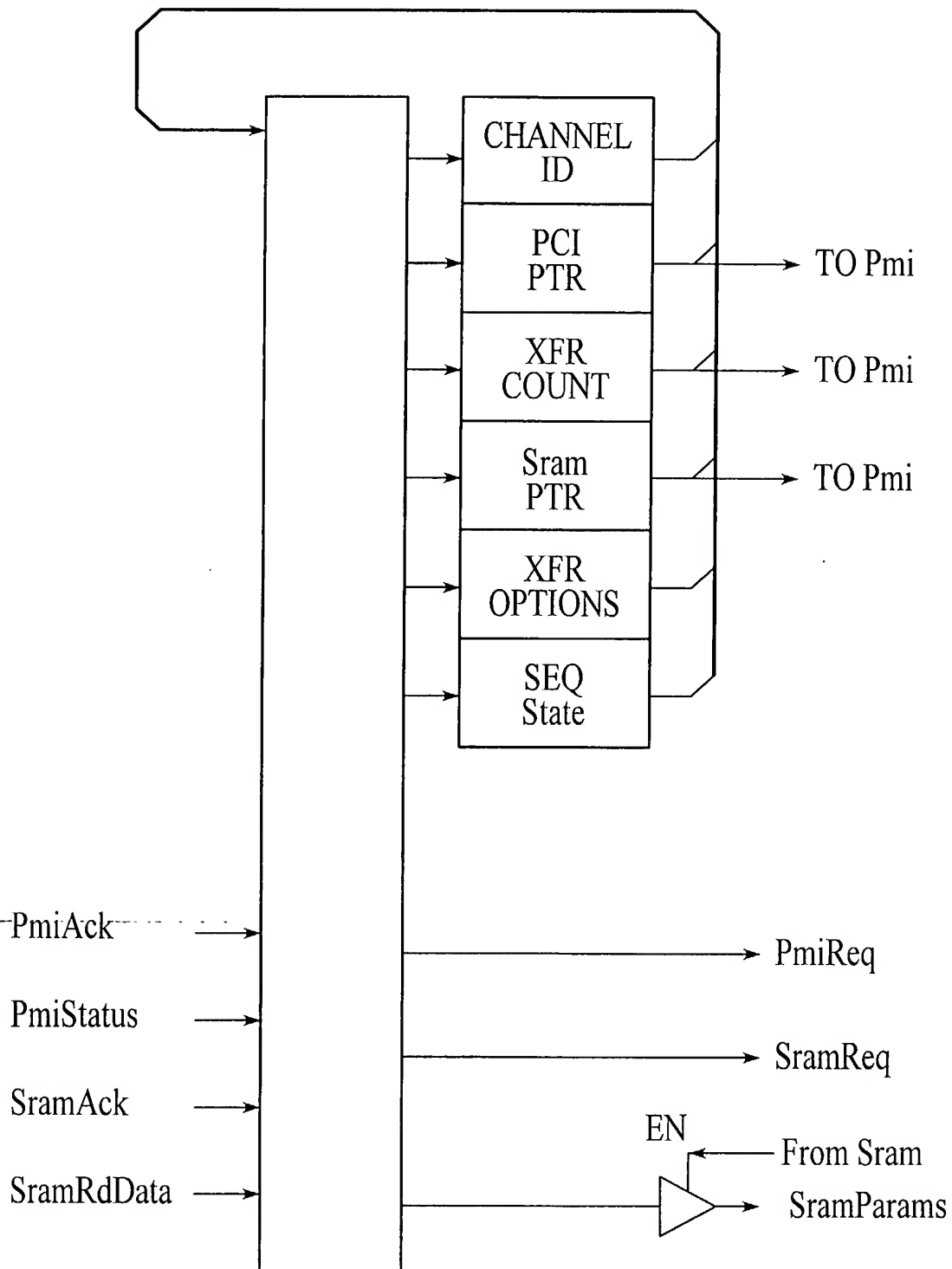


FIG. 71

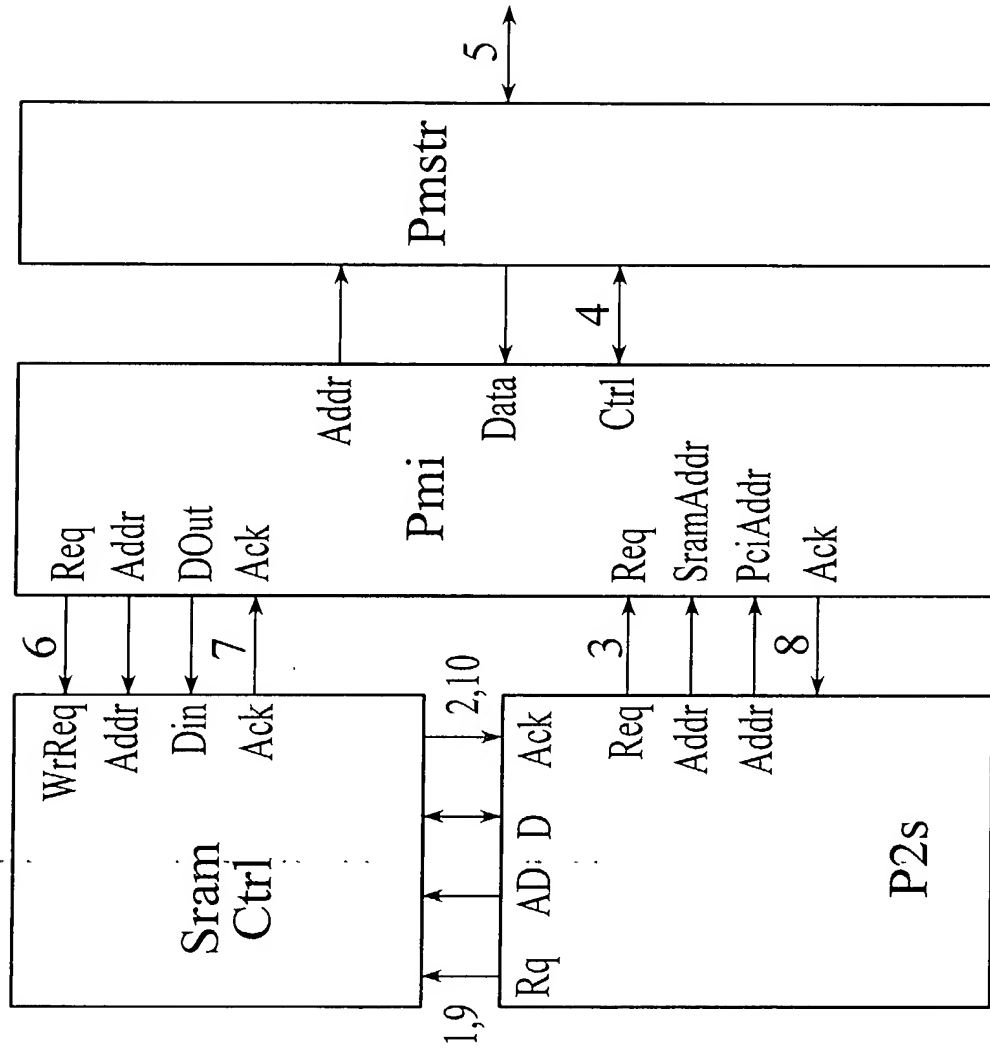


FIG. 72

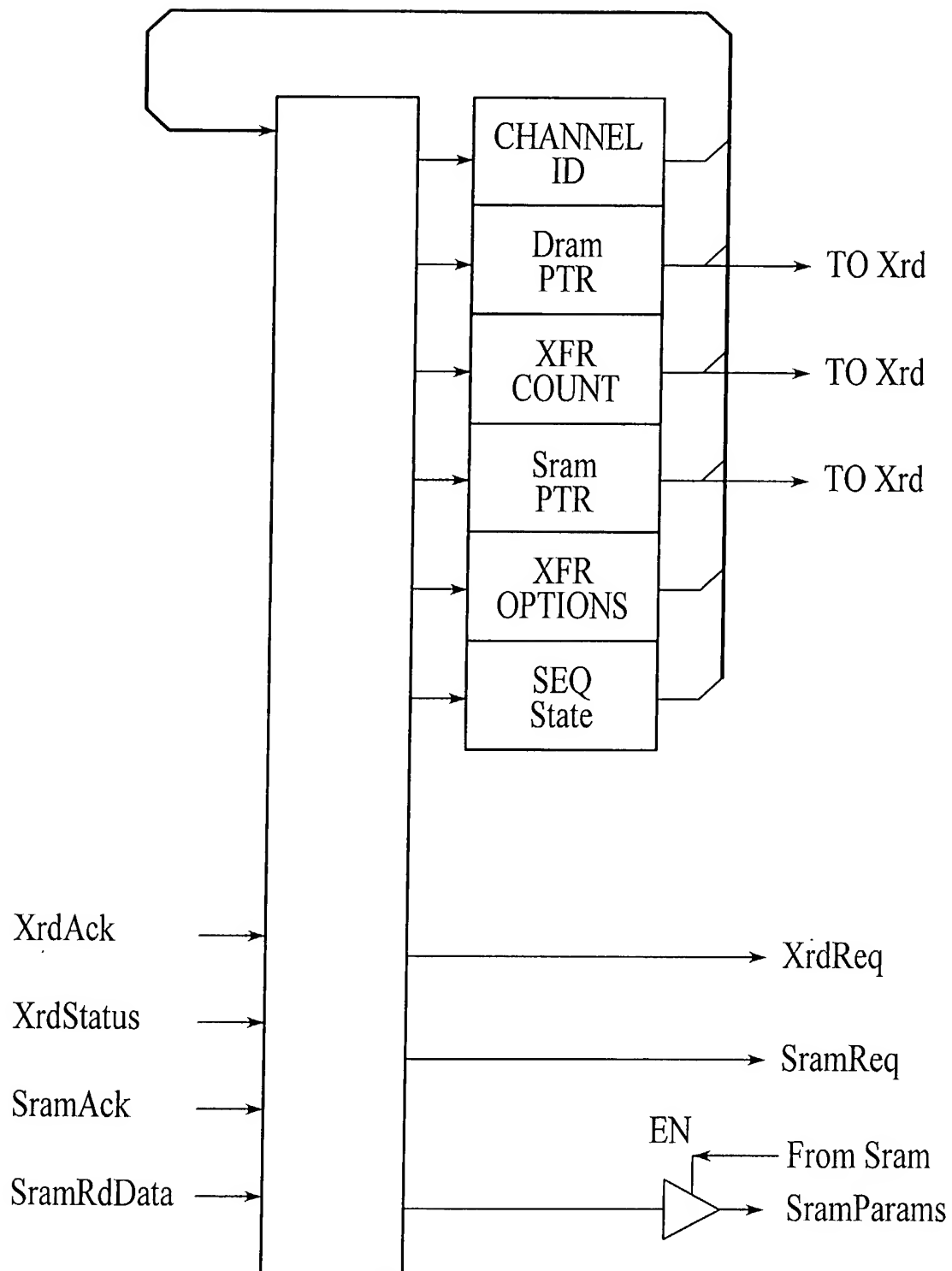


FIG. 73

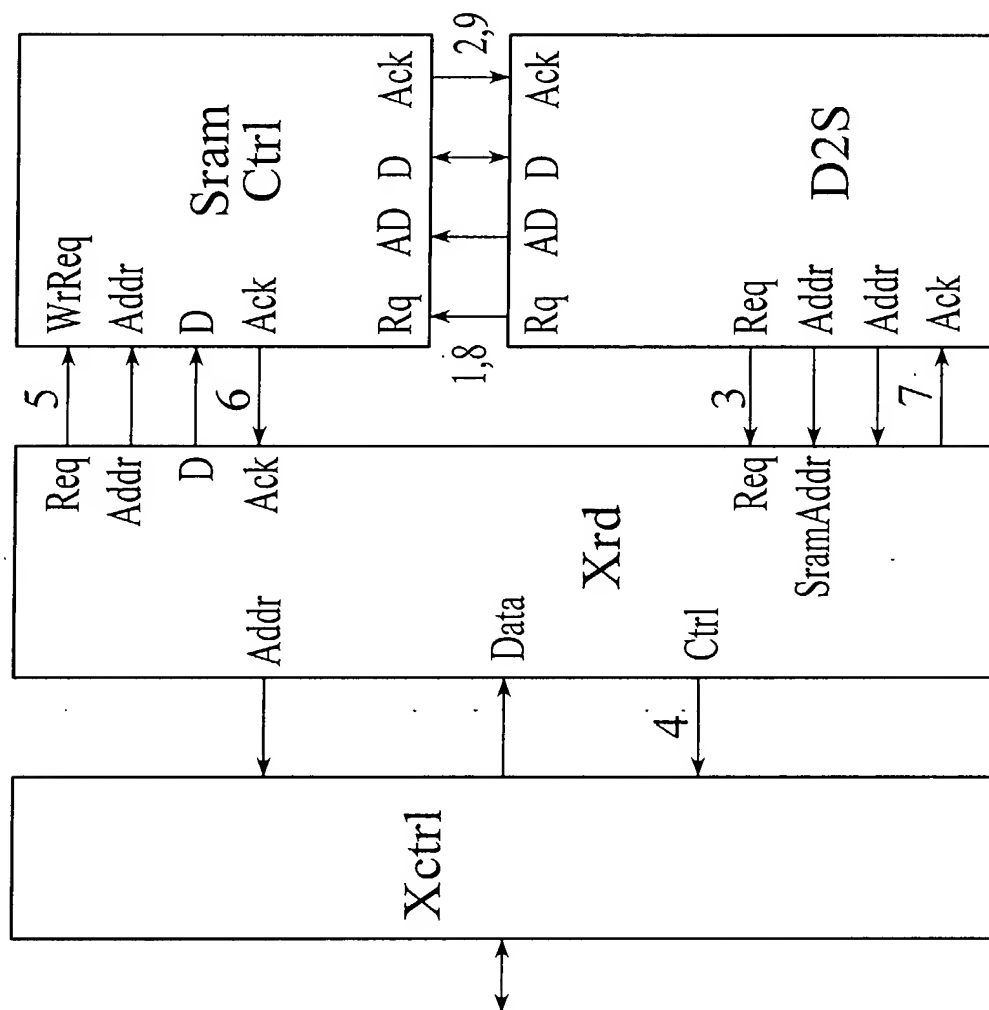


FIG. 74

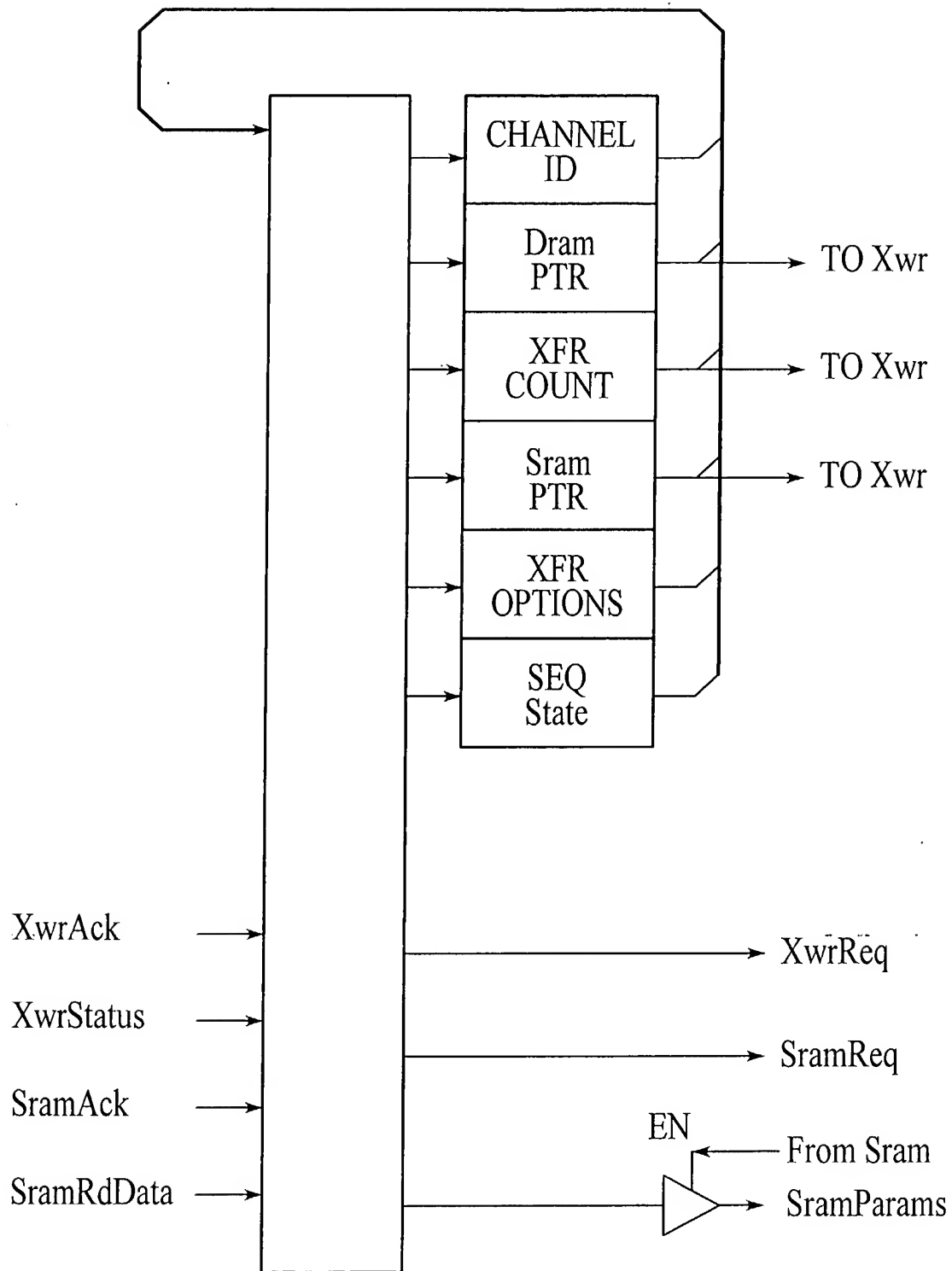


FIG. 75

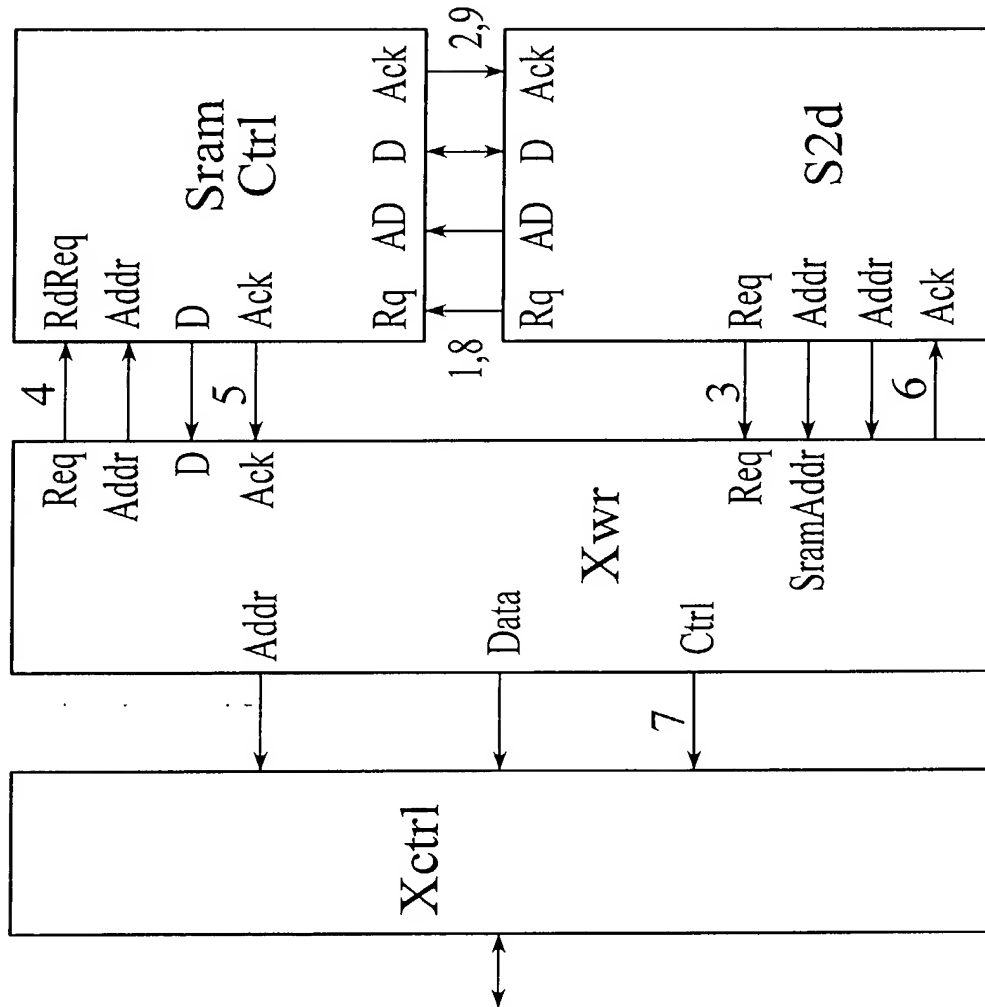


FIG. 76

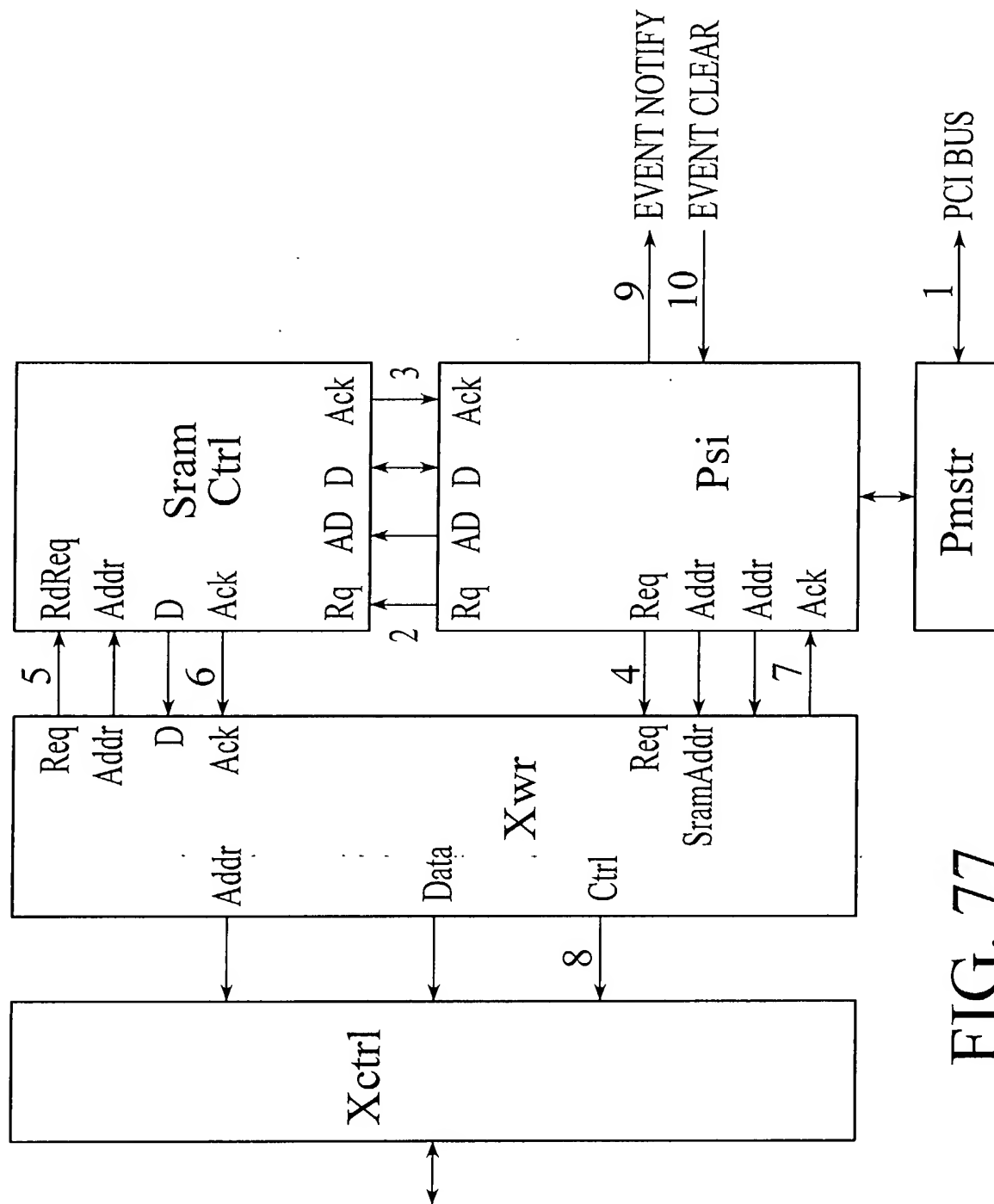


FIG. 77

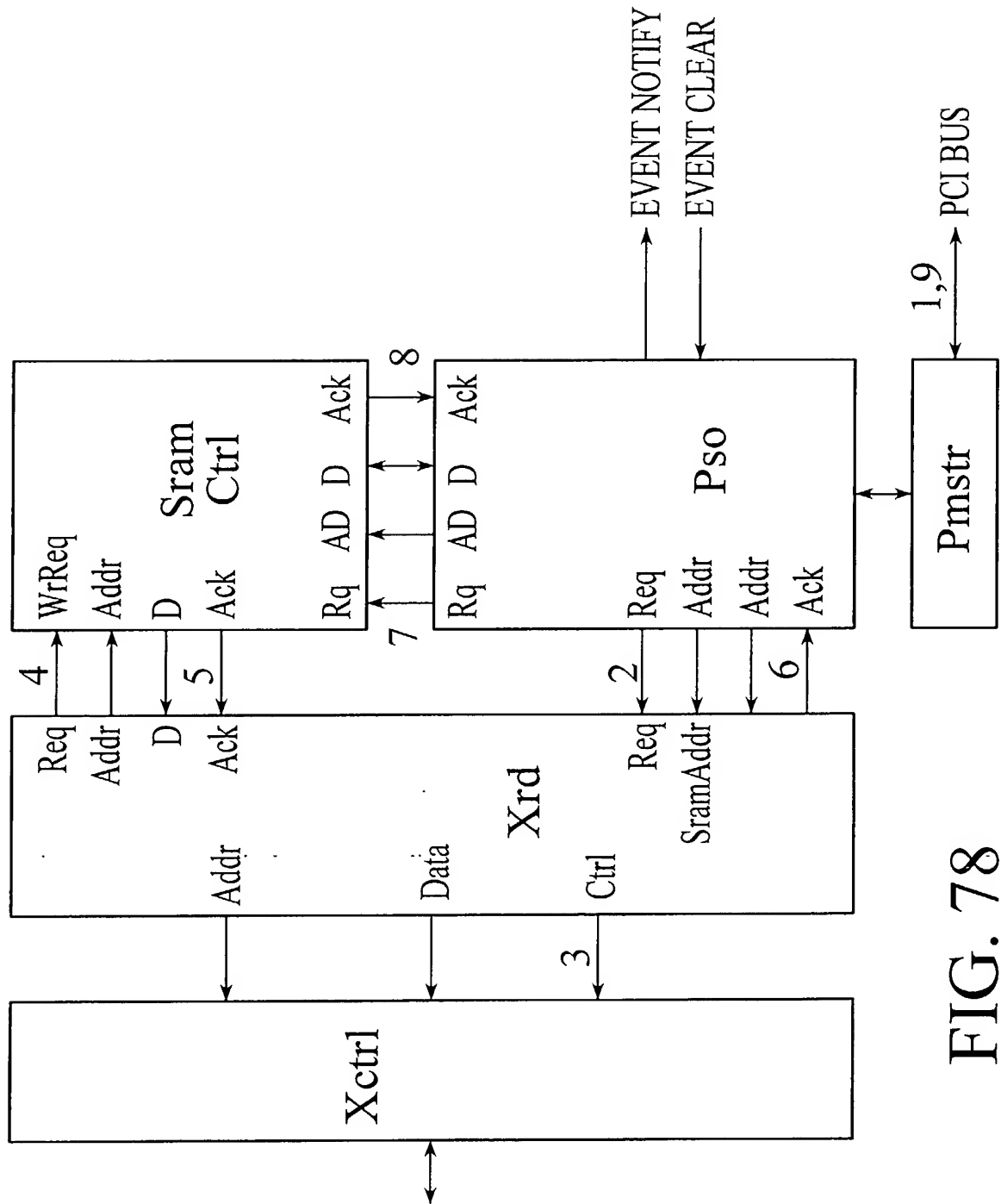
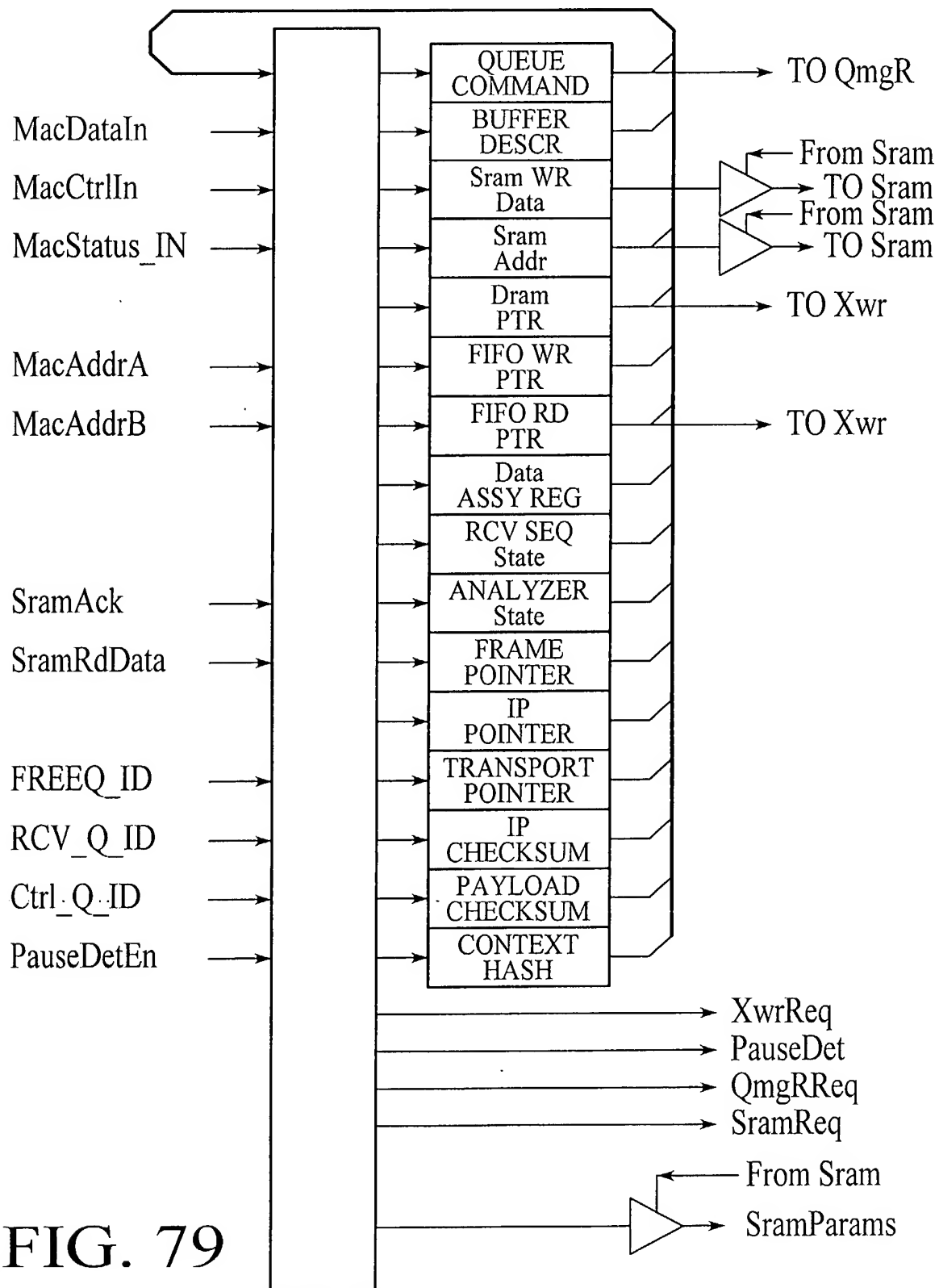


FIG. 78



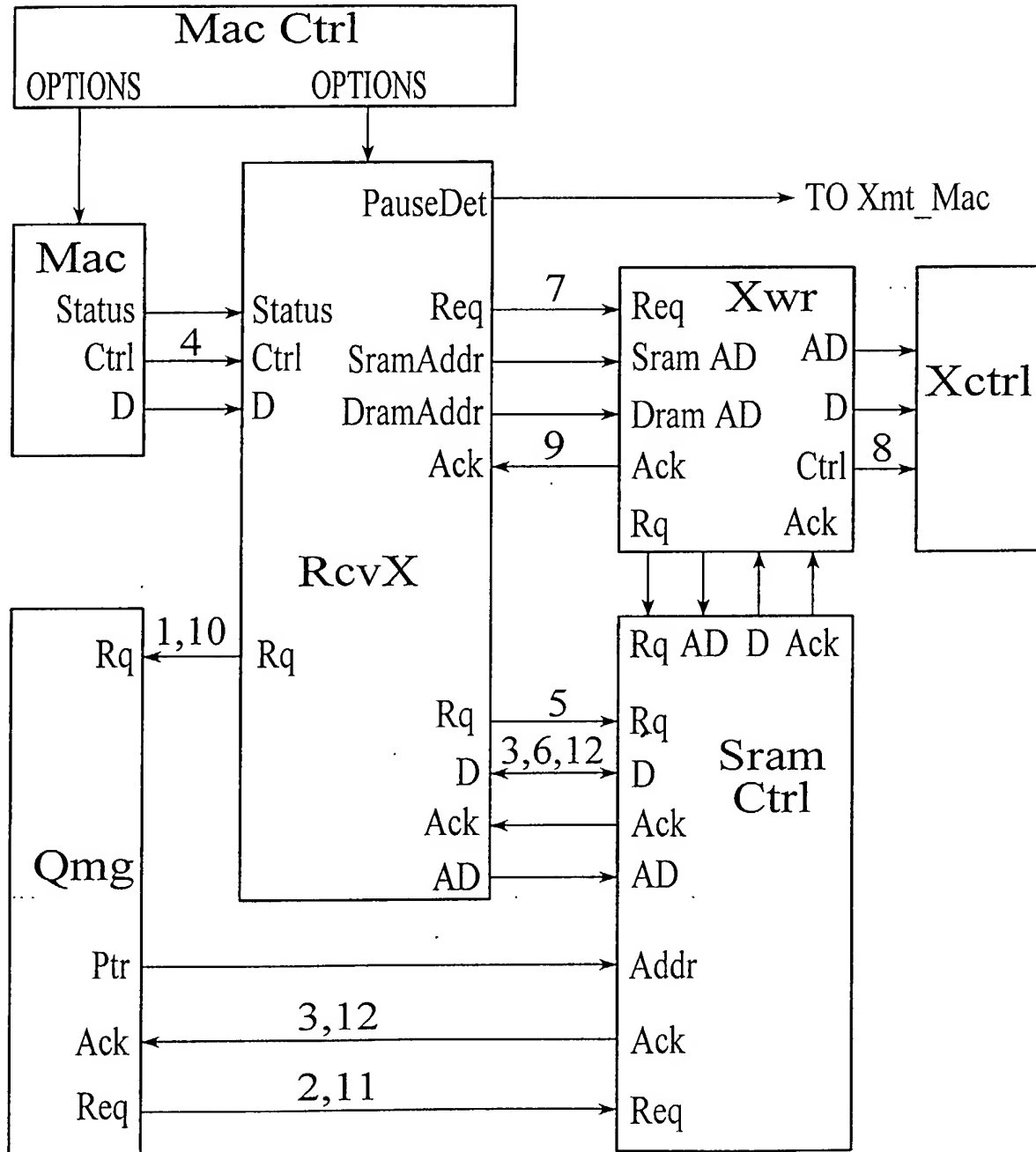


FIG. 80

RECEIVE BUFFER DESCRIPTOR

<u>bit</u>	<u>name</u>	<u>description</u>
31:30	reserved	
29:28	size	A copy of the bits in the FreeBufDscr .
27:00	address	Represents the last address +1 to which frame data was transferred. The address wraps around at the boundary dictated by the S bits. This can be used to determine the size of the frame received.

FIG. 81**TIME STAMP****OFFSET 0x0008:0x000B**

<u>bit</u>	<u>name</u>	<u>description</u>
31:00	RcvTime	The contents of FreeClk at the completion of the frame receive operation.

FIG. 82**CHECKSUM****OFFSET 0x000C:0x000F**

<u>bit</u>	<u>name</u>	<u>description</u>
31:16	IpChksum	Reflects the value of the IP header checksum at frame completion or IP header completion. If an IP datagram was not detected, the checksum provides a total for the entire data portion of the received frame. The data area is defined as those bytes received after the type field of an ethernet frame, the LLC header of an 802.3 frame or the SNAP header of an 802.3-SNAP frame.
15:00	TcpChksum	Reflects the value of the transport checksum at IP completion or frame completion. If IP was detected but session was unknown, the checksum will not include the psuedo-header. If IP was not detected, the checksum will be 0x0000.

RESERVED**OFFSET 0x0010:0x0011****FRAME Data****OFFSET 0x0012:END OF BUFFER****FIG. 83**

RECEIVE BUFFER FORMAT

FRAME Status A OFFSET 0x0000:0x0003

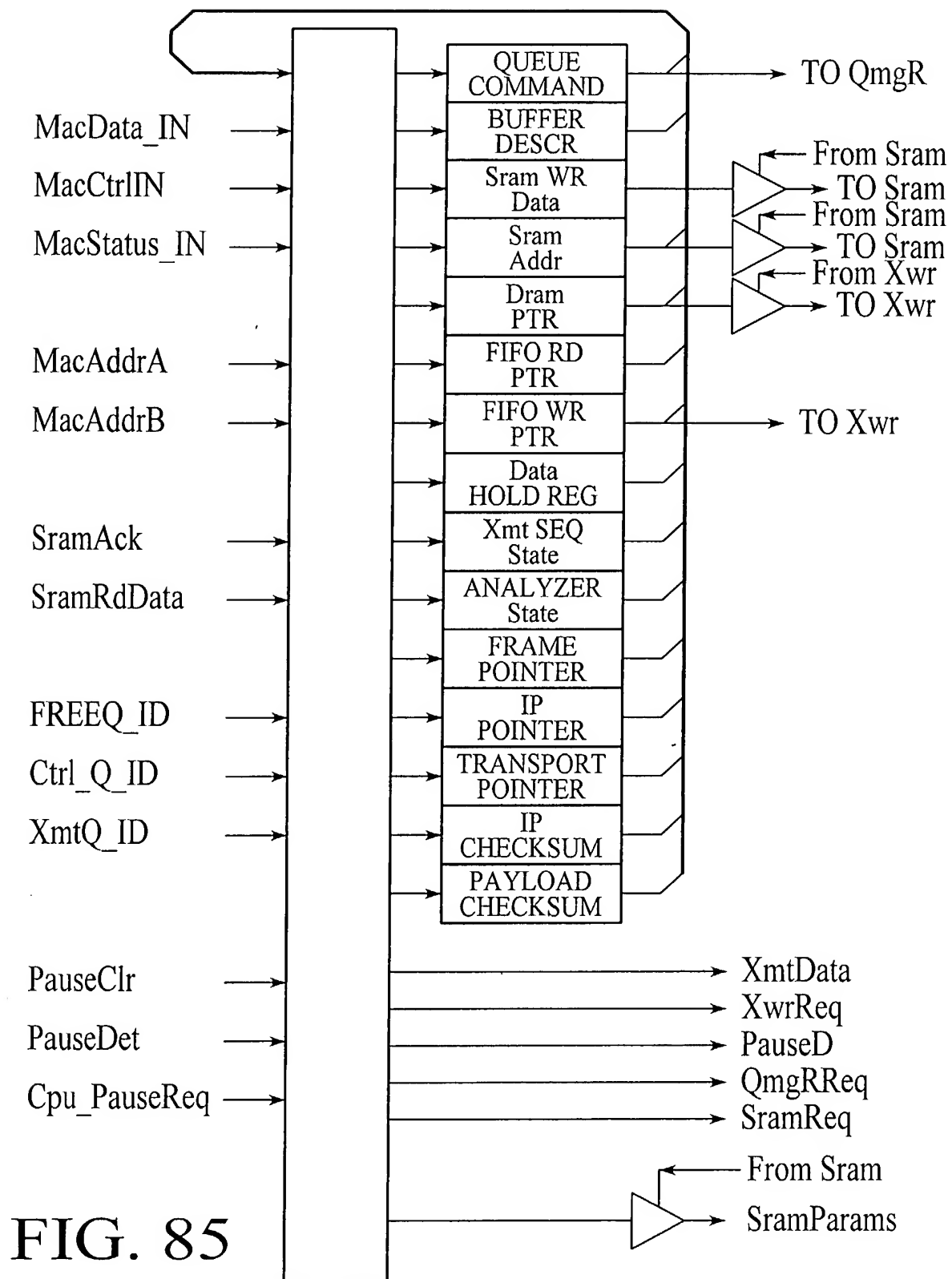
bit	name	description
31	attention	Indicates one or more of the following: CompositeErr , !IpDn , !MacADet & !MacBDet , IpMcast , IpBest , !ethernet & !802.3Snap , !Ip4 , !Tcp .
30	CompositeErr	Set when any of the error bits of ErrStatus are set or if frame processing stops while receiving a Tcp or Udp header.
29	CtrlFrame	A control frame was received at our unicast or special MltCst address.
28	IpDn	Frame processing Hlted due to exhaustion of the IP4 length counter.
27	802.3Dn	Frame processing Hlted due to exhaustion of the 802.3 length counter.
26	MacADet	Frame's destination address matched the contents of MacAddrA .
25	MacBDet	Frame's destination address matched the contents of MacAddrB .
24	MacMcast	The Mac detected a MltCst address.
23	MacBcast	The Mac detected a BrdCst address.
22	IpMcast	The frame processor detected an IP MltCst address.
21	IpBcast	The frame processor detected an IP BrdCst address.
20	Frag	The frame processor detected a Frag IP datagram.
19	IpOffst	The frame processor detected a non-zero IP datagram offset.
18	IpFlgs	The frame processor detected flags within the IP datagram.
17	IpOpts	The frame processor detected a header length greater than 20 for the IP datagram.
16	TcpFlgs	The frame processor detected an abnormal header flag for the TCP segment.
15	TcpOpts	The frame processor detected a header length greater than 20 for the TCP segment.
14	TcpUrg	The frame processor detected a non-zero urgent pointer for the TCP segment.
13	CarrierEvnt	Refer to <i>E110 Technical Manual</i> .
12	LongEvnt	Refer to <i>E110 Technical Manual</i> .
11	FrameLost	Set when an incoming frame could not be processed as a result of an outstanding frame completion event not yet serviced by the utility processor.
10	reserved	
10	NoAck	The frame processor detected a
09:08	FrameTyp	00 - Reserved. 01- ethernet. 10 - 802.3. 11 - 802.3 Snap.
07:06	NwkTyp	00 - Unknown. 01- Ip4. 10 - Ip6 11 - ip other.
05:04	TrnsptTyp	00 - Unknown. 01- reserved. 10 - Tcp 11 - Udp
03	NetBios	A NetBios frame was detected.
02	reserved	
01:00	channel	The Mac on which this frame was received.

FRAME Status B OFFSET 0x0004:0x0007

bit	name	description
31	802.3Shrt	End of frame was encountered before the 802.3 length count was exhausted.
30	BufOvr	The frame length exceded the buffer space available.
29	BadPkt	Refer to <i>E110 Technical Manual</i> .
28	InvldPrmb1	Refer to <i>E110 Technical Manual</i> .
27	CrcErr	Refer to <i>E110 Technical Manual</i> .
26	DrblNbbl	Refer to <i>E110 Technical Manual</i> .
25	CodeErr	Refer to <i>E110 Technical Manual</i> .
24	IpHdrShrt	The IP4 header length field contained a value less than 0x5.
23	IpIncmplt	The frame terminated before the IP length counter was exhausted.
22	IpSumErr	The IP header checksum was not 0xffff at the completion of the IP header read.
21	TcpSumErr	The session checksum was not 0xffff at the termination of session processing.
20	TcpHdrShrt	The TCP header length field contained a value less than 0x5.
19:16	PressCd	The state of the frame processor at the time the frame processing terminated. 0b0000 Processing Mac header. 0b0001 Processing 802.3 LLC header. 0b0010 Processing 802.3 SNAP header. 0b0011 Processing unknown network data. 0b0100 Processing IP header. 0b0101 Processing IP data (unknown transport). 0b0110 Processing transport header (IP data). 0b0111 Processing transport data (IP data). 0b1000 Processing IP processing complete. 0b1001 Reserved. 0b101x Reserved. 0b11xx Reserved.
15:08	MacHsh	The Mac destination-address hash. Refer to <i>E110 Technical Manual</i> .
07:00	CtxHsh	The 8-bit context-hash generated by exclusive-oring all bytes of the IP source address, IP destination-address, transport source port and the transport destination port.

FIG. 84

75/82



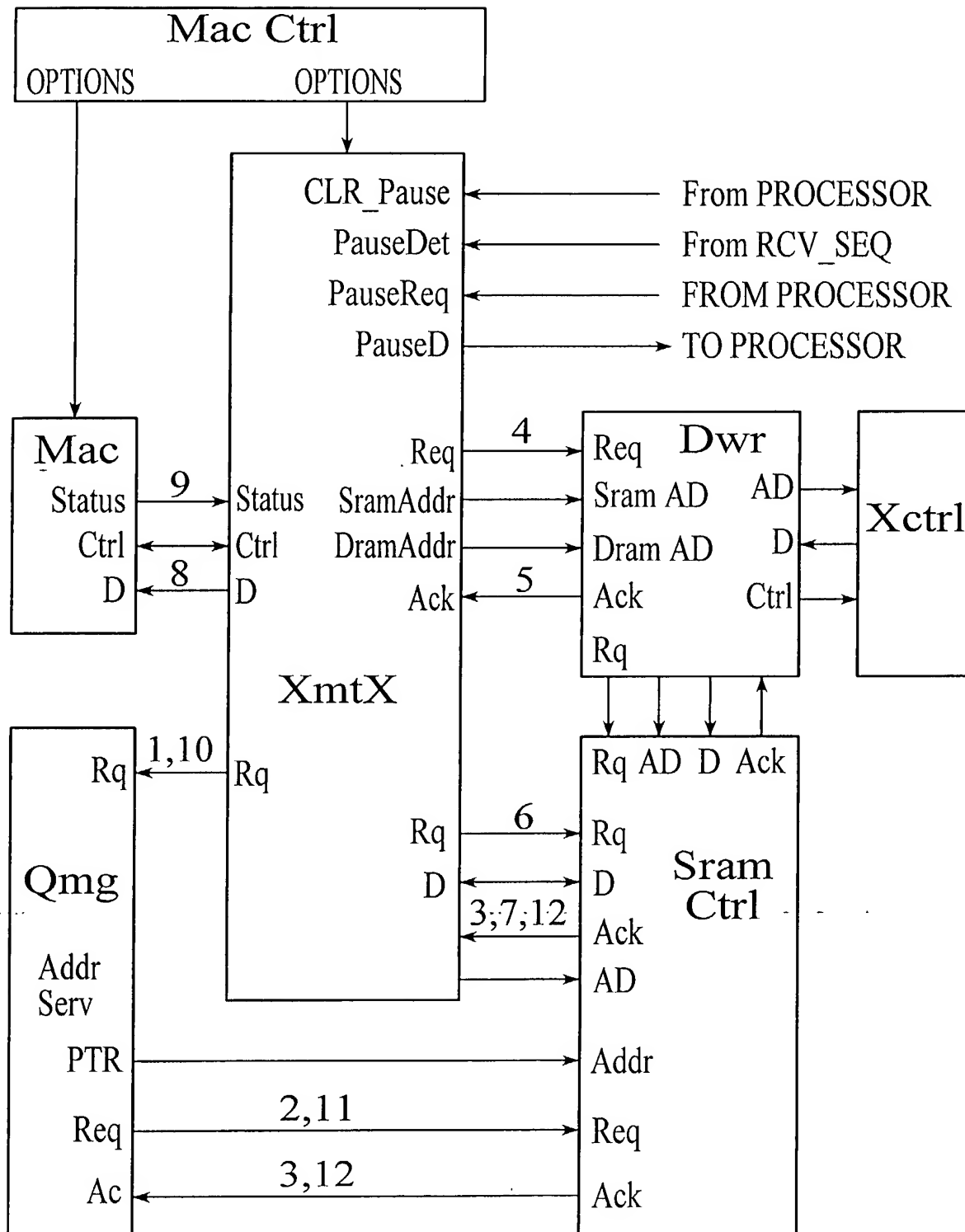


FIG. 86

TRANSMIT BUFFER DESCRIPTOR

bit	name	description
31	ChksumEn	When set, XmtSeq will insert a calculated checksum. When reset, XmtSeq will not alter the outgoing data stream.
30 29:28	reserved size	Represents the size of the buffer by indicating at what boundary the buffer should start and terminate. This is used in combination with EndAddr to determine the starting address of the buffer : S = 0 256B boundary. A[7:0] ignored. S = 1 2KB boundary. A[10:0] ignored. S = 2 4KB boundary. A[11:0] ignored. S = 3 32KB boundary. A[14:0] ignored.
27:00	EndAddr	The address of the last byte to transmit plus one.

FIG. 87

TRANSMIT BUFFER FORMAT

CHECKSUM PRIMER OFFSET 0x0000:0x0003

bit	name	description
31:00	Primer	A value to be added during checksum accumulation. For IPV4, this should include the psuedo-header values, protocol and Tcp-length.

RESERVED OFFSET 0x0004:0x0005

FRAME Data OFFSET 0x0006:END OF BUFFER

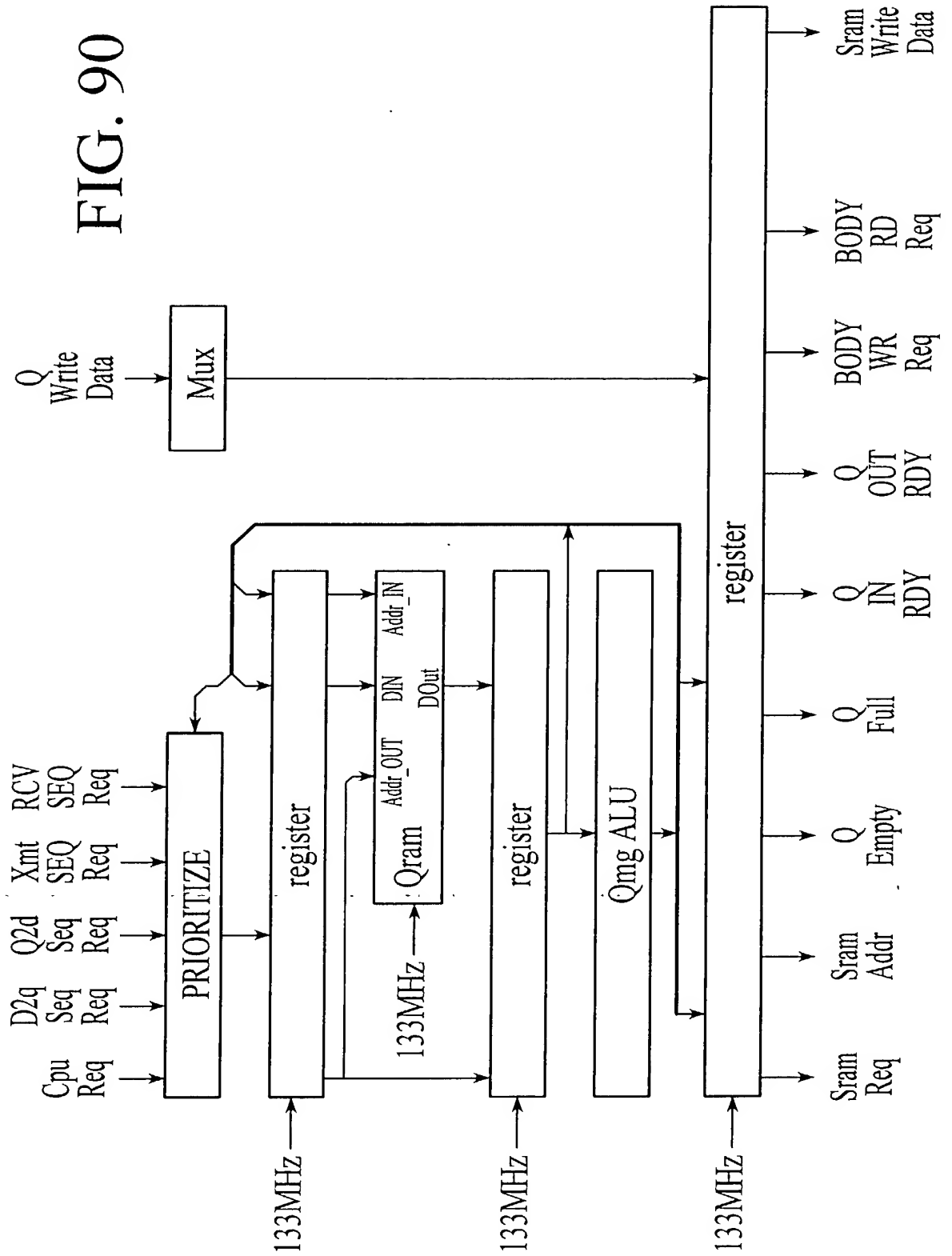
FIG. 88

TRANSMIT Status VECTOR

bit	name	description
31	LnkErr	Indicates that a link status error occurred before or during transmit.
30:15	reserved	
14	ExcessDeferral	Refer to <i>E110 Technical Manual</i> .
13	LateAbort	Refer to <i>E110 Technical Manual</i> .
12	ExcessColl	Refer to <i>E110 Technical Manual</i> .
11	UnderRun	Refer to <i>E110 Technical Manual</i> .
10	ExcessLgth	Refer to <i>E110 Technical Manual</i> .
09	Okay	Refer to <i>E110 Technical Manual</i> .
08	deferred	Refer to <i>E110 Technical Manual</i> .
07	BrdCst	Refer to <i>E110 Technical Manual</i> .
06	MltCst	Refer to <i>E110 Technical Manual</i> .
05	CrcErr	Refer to <i>E110 Technical Manual</i> .
04	LateColl	Refer to <i>E110 Technical Manual</i> .
03:00	CollCnt	Refer to <i>E110 Technical Manual</i> .

FIG. 89

FIG. 90



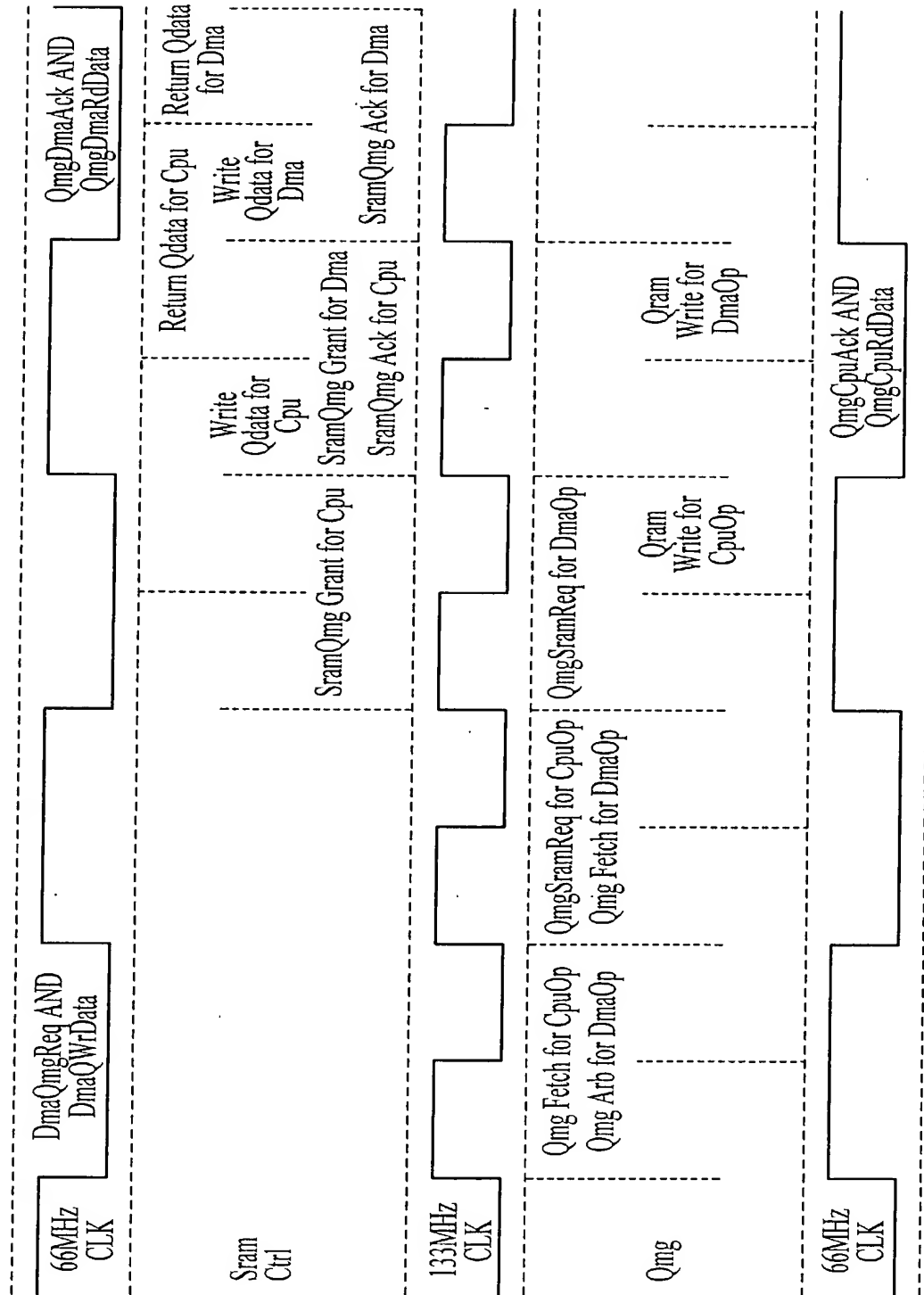


FIG. 91

DMA OPERATIONS

<u>dma seq #</u>	<u>name</u>	<u>description</u>
0	none	This is a no operation address.
1	D2dSeq	Moves data from ExtMem to ExtMem.
2	D2sSeq	Moves data from ExtMem bus to sram.
3	D2pSeq	Moves data from ExtMem to Pci bus.
4	S2dSeq	Moves data from sram to ExtMem.
5	S2pSeq	Moves data from sram to Pci bus.
6	P2dSeq	Moves data from Pci bus to ExtMem.
7	P2sSeq	Moves data from Pci bus to sram.

FIG. 92

<u>bit</u>	<u>name</u>	<u>description</u>
31:11	reserved	Data written to these bits is ignored.
10:8	ChCmd	0 - Stops execution of the current operation and clears the corresponding event flag. 1 - Transfer data from ExtMem to ExtMem. 2 - Transfer data from ExtMem bus to sram. 3 - Transfer data from ExtMem to Pci bus. 4 - Transfer data from sram to ExtMem. 5 - Transfer data from sram to Pci bus. 6 - Transfer data from Pci bus to ExtMem. 7 - Transfer data from Pci bus to Sram.
07:05	reserved	Data written to these bits is ignored.
04:00	ChId	Provides the channel number for the channel command.

FIG. 93

<u>bit</u>	<u>name</u>	<u>description</u>
127:96	PciAddrH	Bits [63:32] of the Pci address.
95:64	PciAddrL	Bits [31:00] of the Pci address.
59:32	MemAddr	Bits [27:00] of the ExtMem address or bits [15:00] of the Sram address.
31	PciEndian	When set, selects big endian mode for Pci transfers.
30	WideDbl	When set, disables Pci 64-bit mode.
22	DstFlash	Selects Flash for the external memory destination of P2d.
15:00	XfrSz	Bits [15:00] of the requested dma size expressed in bytes.

FIG. 94

bit	name	description
123:96	MemAddr	Bits [27:00] of the ExtMem address or bits [15:00] of the Sram address.
95:64	PciAddrH	Bits [63:32] of the Pci address.
63:32	PciAddrL	Bits [31:00] of the Pci address.
30	SrcFlash	Selects Flash for the external memory source of D2p .
23	PciEndian	When set, selects big endian mode for Pci transfers.
22	WideDbl	When set, disables Pci 64-bit mode.
15:00	XfrSz	Bits [15:00] of the requested dma size expressed in bytes.

FIG. 95

bit	name	description
127:124	reserved	Reserved for future use.
123:96	SrcAddr	Bits [27:00] of the ExtMem address or bits [15:00] of the Sram address.
95:60	reserved	Reserved for future use.
59:32	DstAddr	Bits [27:00] of the ExtMem address or bits [15:00] of the Sram address.
30	FlashSel	Selects Flash for the external memory source of D2d or D2s .
22	FlashSel	Selects Flash for the external memory destination of S2p or D2d .
15:00	XfrSz	Bits [15:00] of the requested dma size expressed in bytes.

FIG. 96

bit	name	description
127:64	reserved	Not used.
63:32	ChkSum	Represents the 1's compliment sum of all halfwords transferred during a P2d or D2d operation only.
31:24	reserved	Reserved for future use.
23:20	SrcStatus	TBD.
19:16	DstStatus	TBD.
15:00	XfrSz	Bits [15:00] of the residual dma size expressed in bytes. This value will be zero if the dma operation was successful

FIG. 97

bit	name	description
31:00	ChDn	Each bit represents the done flag for the respective dma channel. These bits are set by a dma sequencer upon completion of the channel command. Cleared when the processor writes 0 to the corresponding ChCmd register ChCmdOp field.

FIG. 98

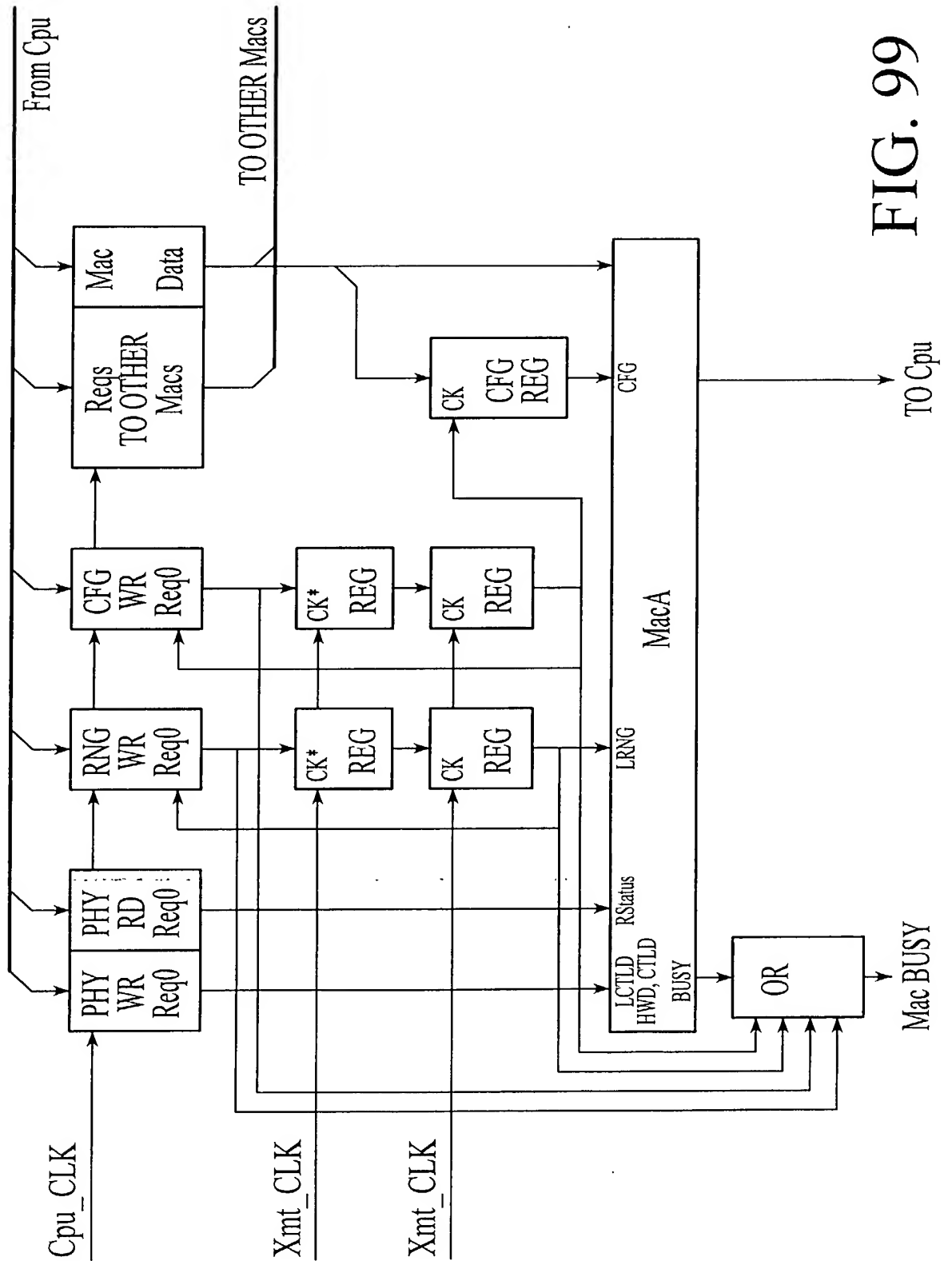


FIG. 99